Universidad Carlos III de Madrid

Aplicación de inicio de sesión mediante autenticación con NFC

Autor: Fermín Gallego de la Sacristana

Tutor: Mario Muñoz

Agradecimientos

A mis compañeros Jorge, María y Jaime, sin su amistad, el camino hubiera sido mucho más difícil.

A mis padres, por haber estado tantos años apoyándome.

A Mario Muñoz y a Gamma Solutions, por haberme dado la oportunidad de trabajar con ellos y haber hecho posible este proyecto.

Resumen

La mayor parte de las aplicaciones NFC han sido desarrolladas para ser utilizadas con teléfonos, bien entre estos o bien con tarjetas NFC. Hasta el momento pocos han sido los lectores que permitieran hacer uso del NFC del mismo modo que un teléfono móvil, y que se facilitara el uso del lector con la disponibilidad de herramientas y kits de desarrollo. El lector ACR122 elegido para este proyecto, ha permitido que se pueda trabajar con la mayor parte de las posibilidades que NFC ofrece. Además se ha aportado una librería y una documentación, que permitirá el desarrollo de futuras aplicaciones basándose en los objetivos que se han logrado en este proyecto.

En cuanto a los teléfonos móviles, desde hace unos años atrás, han sido muchas las pruebas piloto del uso de aplicaciones con NFC. Una de las empresas pioneras fue Nokia, que puso a la venta a pequeña escala terminales con esta característica y facilitando el trabajo con herramientas para desarrolladores para estos dispositivos. En el año actual, ha sido Google quien ha apostado fuerte con esta tecnología incluyéndolo en Android y en nuevos dispositivos que están saliendo al mercado, con intención de completar su plataforma de pagos Google Wallet[12].

No es conveniente hacer uso únicamente de una plataforma móvil, pues actualmente son 3 las principales competidoras del mercado (Android, BlackBerry e iOS) y por ejemplo BlackBerry tiene previsto sacar teléfonos que soporten NFC. Así que cuantas más plataformas móviles se puedan usar para una aplicación, asegura su facilidad de entrar en el mercado y su expansión. En este caso serán 2 las que se utilicen: Android una de las más actuales, y J2ME de las que más tiempo lleva usándose. En la aplicación que aquí se desarrolla se puede observar cómo se puede desarrollar en varias plataformas y cómo pueden convivir conjuntamente.

La aplicación sobre el inicio de sesión o *login* que en este proyecto se desarrolla sirve para visualizar el alcance al que se puede llegar con esta tecnología.

1 Índice

2 Índice de figuras:

3 Índice de tablas:

4 Introducción

4.1 Motivación

Desde hace algunos años, la tecnología NFC ha abierto el camino de nuevas posibilidades en el ámbito de las aplicaciones móviles. Si bien, los tipos de uso que pueden realizarse están bien definidos, aún hay lugar para las nuevas ideas que puedan convertir el NFC en una tecnología que cumpla con unas necesidades que hasta ahora no podían cubrirse de esta manera tan sencilla, segura y a la vez interoperable con otros datos personales que existen dentro de un teléfono móvil.

Hasta la fecha, todas las aplicaciones NFC habían sido realizadas sobre terminales que hacían uso de J2ME. Aunque se trata de una plataforma suficientemente completa y funcional para permitir desarrollar multitud de aplicaciones, queda bastante lejos de las posibilidades que ofrecen los actuales smartphones y sus sistemas operativos como Android, que ofrecen una plataforma mucho más completa, hacer un uso mucho más completo del teléfono como pueden ser datos de carácter personal, vías de comunicación, sincronización en la nube, etc.

Google hizo su apuesta por NFC en Android en el actual año 2011, con su versión Gingerbread en el Samsung Nexus S. En un principio fue utilizada para casos de uso sencillo, como son el intercambio de tarjetas personales, pero actualmente, ya se están realizando las primeras aplicaciones de pago, una vez Google ha perfeccionado su plataforma de pagos Google Wallet[12].

Además la mayor parte de las aplicaciones han sido desarrolladas para utilizarse únicamente entre teléfonos móviles. La dificultad de utilizar un PC estriba en la falta de lectores suficientemente compatibles con los diferentes protocolos NFC, la escasez de librerías suficientemente desarrolladas, documentadas y de fácil acceso (si bien en estos detalles el proyecto *libnfc*[4] ha avanzado considerablemente).

Así que actualmente, resultaba bastante interesante realizar un proyecto que permitiera realizar comunicaciones NFC entre móviles y PC de tal forma que sirviera como referencia para futuras aplicaciones de diversa índole.

4.2 Objetivos

Una vez han quedado claro las motivaciones que llevan a hacer este proyecto dentro del contexto actual, se muestran los objetivos que va a cumplir el presente proyecto:

1. Analizar y documentar los diferentes elementos que participan en las comunicaciones NFC (teléfonos, tarjetas, lectores), considerando las opciones que ofrece cada tipo.
2. Realizar una librería suficientemente completa, estable y funcional, que permita ser utilizada y/o adaptada para los diferentes casos de uso.

3. Mostrar el alcance y potencial de la librería desarrollando una aplicación de autenticación e inicio de sesión.
4. Integrar dentro del sistema operativo la aplicación desarrollada.

4.3 Contenido del documento

En este apartado se realiza una breve descripción del contenido de los diferentes apartados, que pueda servir como resumen o guía para acceder a los mismos.

- Tabla de contenido: Muestra el índice de capítulos.
- Índice de figuras: Muestra donde se sitúan las imágenes.
- Introducción: Breve explicación del proyecto.
- Estado del arte: Exposición de temas relacionados con el proyecto.
- Aplicación de login: Aplicación realizada. Se divide en varias partes que comprenden el desarrollo de la librería, la aplicación para el PC y la aplicación para teléfonos móviles.
- Conclusiones: Consideraciones finales tras haber terminado el proyecto y trabajos futuros.
- Diagrama de Gantt: Diagrama con las fases y duración tentativa.
- Presupuestos: Costes estimados para la realización obtenida
- Bibliografía.
- Anexos.

5 Estado del arte

5.1 NFC

Near Field Communication (NFC) es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos a menos de 10cm, en el que uno de los cuales puede funcionar de forma pasiva (tarjetas sin alimentación).

Funciona en la banda de 16.56 MHz, la cual es una frecuencia libre que se puede usar sin licencias con distintas velocidades de 106kbit/s, 212kbit/s o 424kbit/s, según soporten los dispositivos.

NFC es una evolución del anterior RFID (ISO 14443). Un dispositivo NFC puede comunicarse con cualquier tarjeta inteligente y lector, existentes dentro del estándar ISO/IEC 14443.La principal diferencia es que el alcance es mucho menor, por lo que se necesita que los dispositivos a interaccionar estén a escasos centímetros durante un instante para la transmisión de información. Aunque esta característica parezca una limitación, más bien es una ventaja de esta tecnología, ya que por el hecho de que un dispositivo este suficientemente próximo, podemos dar por confirmado que el usuario es consciente del acto que está haciendo, reflejando la voluntad de utilizar dichos servicios y de que no se le va a dar un uso fraudulento. Además de tratarse de un método muy cómodo e intuitivo para el usuario.

Su desarrollo empieza en el año 2002 y sus promotores fueron Philips y Sony principalmente para conseguir compatibilidad con sus tecnologías, Mifare y FeliCa respectivamente, pero no sino hasta finales del año 2003 que se la aprueba como el estándar ISO 18092 y la ECMA-340.

Estos estándares especifican los esquemas de modulación, codificación, velocidades de transferencia y formato de la trama de la interfaz RF de dispositivos NFC, así como los esquemas de inicialización y condiciones requeridas para el control de colisión de datos durante la inicialización para ambos modos de comunicación, activo y pasivo. También definen el protocolo de transporte, incluyendo los métodos de activación de protocolo y de intercambio de datos.

NFC incorpora una variedad de estándares pre-existentes incluyendo ISO/IEC 14443 de ambos tipos, tipo A y tipo B. Por lo tanto los teléfonos habilitados para NFC muestran interoperabilidad básica con módulos que ya existen como RFID.

NFC no está diseñada para la transferencia masiva de información, pero su punto fuerte es que puede ser utilizada para iniciar y configurar el acceso a otro tipo de redes inalámbricas de mayor ancho de banda como puede tratarse de Bluetooth o Wi-Fi. De esta manera es mucho más cómodo para el usuario, la aproximación del teléfono móvil a un punto en concreto, que introducir claves y contraseñas.

Ilustración 1: Logo de NFC

5.1.1 Funciones

Las posibilidades que ofrece la tecnología NFC permiten que se adopte en multitud de utilidades[14] bien complementándolas o bien dando un nuevo enfoque a esa función:

- **Pago y comercio electrónico**[13]**:** Hasta la fecha no se ha logrado implantar con éxito un sistema de pagos con el teléfono móvil. La tendencia actual es la posibilidad de realizar cualquier operación con el teléfono, además de tener disponible y sincronizados datos de carácter personal, multimedia, agendas, etc. Aún es necesario dar el paso para poder realizar compras en establecimientos, de una forma sustitutiva o complementaria a las actuales tarjetas de crédito o débito. En este ámbito la tecnología NFC ofrece una solución para este tipo de gestiones.
- **Identificación:** El DNI electrónico[9] ha permitido la posibilidad de realizar todo tipo de trámites vía web con un lector. En la línea de integración de todo tipo de servicios en el teléfono móvil, éste es un paso que aún no se ha dado. Facilitará la tramitación de gestiones personales, así como permitirá la incorporación de diversas mejoras, como la integración de diferentes documentos identificativos (Carnet de conducir, Seguridad Social) u otra información personal complementaria (como información médica importante).
- **Fidelización y cupones descuento (loyalty & cuponing):** La sustitución de sustitución de los cupones descuento de papel, por nuevas formas de enviar y aceptar ofertas de forma digital. Las tarjetas de fidelización integradas en el teléfono móvil permiten premiar al consumidor de una forma más personalizada (teniendo en cuenta sus preferencias y hábitos de compra) e instantánea.
- **Control de acceso:** La idea principal es el uso de tarjetas o teléfonos NFC como llaves electrónicas. Implementado en un edificio, se puede saber quién y cuando

accedió una persona, los lugares a los que accedió, así como tener diferentes permisos para diferentes usuarios. Estos permisos son fácilmente modificables en caso de tenerlos de forma centralizada.

- **Ticket electrónico:** Para usarlo como billete en servicios de transporte público [11], entradas para todo tipo de situaciones como pueden ser conciertos, eventos deportivos, cines, museos, aparcamientos.
- **Conexiones de red:** Iniciar conexiones con otro tipo de comunicaciones, que requieren algún tipo de autorización o información previa: emparejamiento entre dispositivos con Bluetooth, o acceso a una red Wi-Fi protegida[10].
- **Tarjetas de presentación:** En una etiqueta NFC es fácil compartir información de contacto (nombre, e-mail, número de teléfono,…). En este aspecto tiene como competencia los códigos QR.
- **Inventariado:** Las etiquetas NFC permiten realizar una identificación y clasificación de productos, de igual manera que un código de barras. Para este tipo de aplicaciones es más frecuente utilizar RFID, ya que al funcionar a una distancia mayor, es más cómodo de utilizar.
- **Check-in:** Confirmar que realmente se está en un determinado lugar o establecimiento y poder compartirlo.

5.1.2 Ventajas

A modo de resumen, las ventajas que ofrece utilizar aplicaciones NFC son las siguientes:

- **Proximidad**: Esta característica hace más seguro e intuitivo su uso, pues solo se realiza la correspondiente operación (pagos, identificación,…), si el usuario es consciente y tiene verdadera intención de ello. Además dificulta que exista un tercero que observe la información transmitida.
- **Seguridad**: La mayor parte de las aplicaciones se ejecutan dentro de un entorno seguro (Elemento Seguro o Secure Element) que se encuentra dentro del teléfono o la tarjeta SIM. Para hacer uso de éste, se necesitan aplicaciones firmadas, lo que dificulta y casi imposibilita el uso de las aplicaciones con fines fraudulentos.
- **Paso al modo electrónico:** Funciones como el pago, el acceso a lugares con llaves convencionales pasa a ser parte de esta tecnología, que a partir de ahora evoluciona para poder integrarse e interactuar con otras utilidades.
- **Todo en uno:** La tendencia actual es a poder realizar todo tipo de gestiones desde un smartphone. La inclusión del NFC dentro de los terminales incrementa el número de situaciones en las que puede resultar útil tener todas las aplicaciones en un mismo dispositivo y que interactúan entre ellas(por ejemplo, realizar check-in de un evento o lugar y compartirlo en una red social).

5.1.3 Establecimiento de la conexión

La conexión NFC se produce en 5 etapas:

- Descubrimiento: En esta fase los dispositivos inician la etapa de rastrearse el uno al otro y posteriormente su reconocimiento.
- Autenticación: En esta parte los dispositivos verifican si el otro dispositivo está autorizado o si deben establecer algún tipo de cifrado para la comunicación.
- Negociación: En esta parte del establecimiento, los dispositivos definen parámetros como la velocidad de transmisión, la identificación del dispositivo, el tipo de aplicación, su tamaño, y si es el caso también definen la acción a ser solicitada.
- Transferencia: Una vez negociados los parámetros para la comunicación, se puede decir que ya está realizada exitosamente la comunicación y ya se puede realizar el intercambio de datos.
- Confirmación: El dispositivo receptor confirma el establecimiento de la comunicación y la transferencia de datos.

5.1.4 Arquitectura NFC

En la Ilustración 2 se puede observar las tres diferentes configuraciones en las que esta tecnología puede trabajar:

- Modo Emulación de Tarjeta Inteligente NFC
- Modo de Comunicación Peer-to-Peer
- Modo Lectura / Escritura

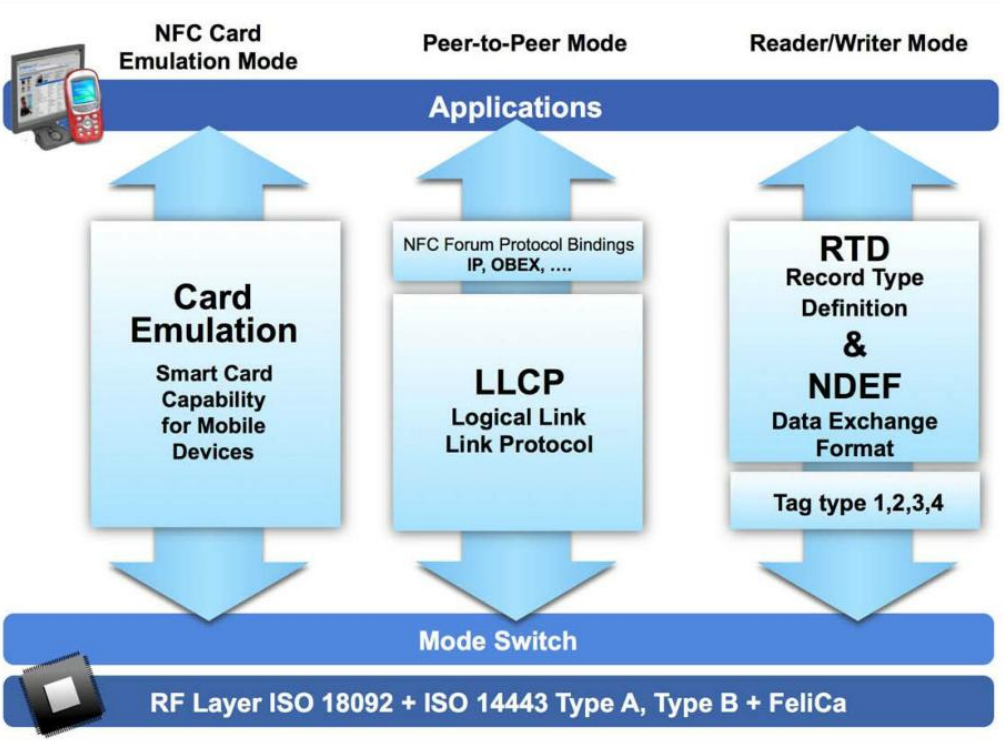


Ilustración 2: Modos de trabajo NFC[1]

La Capa de Radio Frecuencia en la que NFC trabaja está definida en los estándares ISO/IEC 18092 / ECMA – 340: NFCIP-1 e ISO/IEC 21481 / ECMA – 352: NFCIP-2; así como también es compatible con tecnologías ya existentes definidas en la ISO/IEC 14443 en ambos tipos, tipo A y tipo B, al igual que FeliCa.

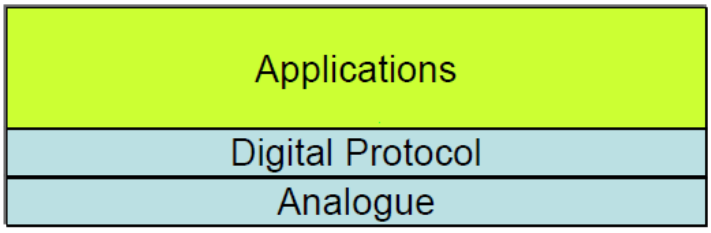


Ilustración 3: Card Emulation Mode[2]

[1] Imagen obtenida de:[19] Arquitectura NFC
NFC Technology Overview, Jonathan Main (September 2009)

[2] Imagen obtenida de:[19] Arquitectura NFC
NFC Technology Overview, Jonathan Main (September 2009)

Modo Emulación de tarjeta inteligente: Este modo se utiliza para que el dispositivo NFC actúe como una etiqueta o una tarjeta inteligente. En este modo también se puede utilizar las características de seguridad avanzadas del elemento seguro, siendo útil para transacciones bancarias por ejemplo o para la gestión de entradas, en general para las gestiones de pagos rápidos, control de accesos, etc.

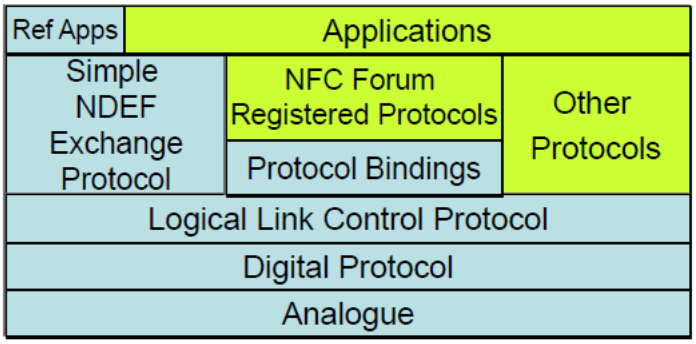


Ilustración 4: Peer-to-Peer Mode[3]

Modo Peer-to-Peer: Este modo sirve básicamente para el intercambio de pequeñas cantidades de datos utilizando el mismo protocolo de NFC. Pero si es necesario un intercambio de una mayor cantidad de información, la comunicación NFC se podría utilizar para establecer parámetros de una comunicación inalámbrica como Bluetooth o Wi-Fi.

NFC utiliza a nivel de la capa de enlace el protocolo de control de enlace lógico (LLCP), el mismo que es usado para la activación, supervisión y desactivación de la comunicación. El modo de transferencia se lo hace de modo asincrónico balanceado, es decir que cualquier dispositivo puede iniciar la transmisión sin permiso de la otra.

El protocolo de intercambio simple NDEF se utiliza para enviar mensajes con el formato NDEF en el modo Peer-to-Peer, al igual que en las especificaciones de operación de los tipos de etiquetas NFC.

El Protocolo de conexión (*Protocolo Bindings*) proporciona enlaces estándar para protocolos NFC registrados y permite su uso interoperable.

Los Protocolos NFC registrados son aquellos que el NFC Forum define un enlace para el Protocolo de Control de Enlace Lógico, por ejemplo IP, OBEX, etc.

Las aplicaciones en modo Peer-to-Peer podrían ser por ejemplo intercambiar imágenes o iniciar una conexión Bluetooth.

[3] Imagen obtenida de:[19] Arquitectura NFC
NFC Technology Overview, Jonathan Main (September 2009)

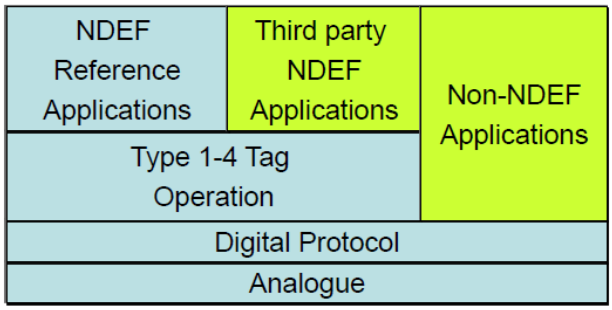


Ilustración 5:Reader/Writer Mode[4]

Modo Lectura / Escritura: En este modo el dispositivo NFC puede leer los cuatro tipos de tarjetas NFC definidos en el *NFC Forum*. Cuando se establece esta configuración los dispositivos NFC pueden intercambiar pequeñas cantidades de información como por ejemplo una información de texto en claro, una dirección web o un número telefónico.

5.1.5 Tipos de tarjetas

Tarjetas Mifare1K

Mifare es una tecnología propiedad de *NXP Semiconductors*, mayoritariamente utilizada en tarjetas inteligentes sin contacto (TISC) y tarjetas de proximidad. Esta tecnología utiliza un protocolo propietario por encima de ISO 144443-A. Las tarjetas Mifare Classick 1k son tarjetas de memoria protegida, divida en sectores y bloques con mecanismo de seguridad personalizable para el control de acceso. Su capacidad de cómputo no permite realizar operaciones criptográficas o de autentificación de alto nivel. Gracias a su fiabilidad y bajo coste, las tarjetas son ampliamente utilizadas como monederos electrónicos, control de acceso, tarjetas de identidad corporativa, tarjetas de transporte urbano o entradas a eventos.

[4] Imagen obtenida de: Arquitectura NFC,NFC Technology Overview, Jonathan Main (September 2009)

		Byte Number within a Block														
Sector	**Block**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Description**
15	3	Key A						Access Bits				Key B						Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A						Access Bits				Key B						Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	
:	:																	
:	:																	
1	3	Key A						Access Bits				Key B						Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0																	Manufacturer Block

Ilustración 6: Esquema Mifare1k[5]

Las Mifare Classic 1K tienen una capacidad de 1024 bytes divididos en 16 sectores. A su vez, cada sector está dividido en 4 bloques de 16 bytes, 3 de los cuales se utilizan para guardar la información de usuario de forma libre, fácilmente modificables con comandos de lectura o escritura. En el bloque restante de cada sector se registran las claves A y B y los permisos de acceso de ese sector. Los permisos pueden ser de lectura, escritura incremento o decremento. Estas 2 tipos de operaciones facilitan el uso de una Mifare Classic como monedero virtual. Además se puede definir una clave de lectura y otra de escritura, ambas claves para ambas funciones o permitir únicamente la lectura con una de las claves.Como último detalle añadir que el primer bloque del primer sector se reserva para detalles del fabricante. Además el primer sector se suele utilizar para organizar las diferentes aplicaciones que hacen uso de cada sector.

Mifare Ultraligth Las Mifare Ultralight son de menor coste, debido a su menor capacidad y a que no incluyen cifrado. La memoria total es de 64 bytes en sectores de 4 bytes. Los primeros 16 bytes se utilizan para información del fabricante, dejando 48

[5] Imagen obtenida de: [17] Tarjetas Mifare 1K (Philips Semiconductors) Mifare 1k Standard Card IC MF1 IC S50 (Rev. 1.5 – May 2001)

bytes libres para poder escribir o leer datos. Estos 48 bytes coinciden con la cantidad de información disponible en un sector de una Mifare 1k (16 bytesx3=48 bytes).

Byte Number	0	1	2	3	Page
Serial Number	SN0	SN1	SN2	BCC0	0
Serial Number	SN3	SN4	SN5	SN6	1
Internal / Lock	BCC1	Internal	Lock0	Lock1	2
OTP	OTP0	OTP1	OTP2	OTP3	3
Data read/write	Data0	Data1	Data2	Data3	4
Data read/write	Data4	Data5	Data6	Data7	5
Data read/write	Data8	Data9	Data10	Data11	6
Data read/write	Data12	Data13	Data14	Data15	7
Data read/write	Data16	Data17	Data18	Data19	8
Data read/write	Data20	Data21	Data22	Data23	9
Data read/write	Data24	Data25	Data26	Data27	10
Data read/write	Data28	Data29	Data30	Data31	11
Data read/write	Data32	Data33	Data34	Data35	12
Data read/write	Data36	Data37	Data38	Data39	13
Data read/write	Data40	Data41	Data42	Data43	14
Data read/write	Data44	Data45	Data46	Data47	15

Ilustración 7: Esquema Mifare Ultraligth[6]

Estas tarjetas también pueden ser utilizadas como tarjetas de transporte o entradas para eventos, pero evidentemente de forma mucho menos segura.

5.1.6 Protocolo NDEF[7]

NFC Forum ha definido un formato de datos común llamado NDEF[20], por sus siglas en inglés NFC Data Exchange Format, el cual puede ser usado para guardar y transportar diferentes tipos de elementos, que van desde cualquier objeto escrito MIME hasta documentos RTD ultra pequeños, tales como URLs.

[6] Imagen obtenida de:[18] Mifare Ultraligth (Philips Semiconductors)
Contactless Single-trip Ticket IC MF0 IC U1 (Rev 3.0 – March 2003)

[7] Texto extraído del documento:[25] Diseño e implementación de un prototipo para control de acceso de personas aplicando la tecnología NFC por medio del uso de teléfonos celulares compatibles con esta tecnología.
Escuela Politécnica Nacional (Ecuador)
Autores: Veloz Chérrez, Diego Fernando

La especificación NDEF define un formato de encapsulación de mensaje para el intercambio de datos entre dispositivos NFC o de un dispositivo NFC a una etiqueta NFC y las reglas para construcción de un mensaje NDEF válido y también de una cadena ordenada de registros NDEF. La diferencia entre una etiqueta y un dispositivo NFC es que la primera no permite una interacción con el usuario y por sí sola no podría mostrar ninguna información al usuario, además es pasiva es decir que no genera su propia energía de funcionamiento y necesita de un dispositivo activo para que funcione. En cambio un dispositivo NFC permite una interacción del usuario así como es el propio generador de su energía y a través de su campo de inducción puede estimular y generar la energía para el funcionamiento de los elementos pasivos.

NDEF es un formato binario ligero que puede encapsular uno o más payloads de diferente tipo y tamaño dentro de la estructura de un solo mensaje. El payload está identificado por un tipo, una longitud y un identificador opcional.

- Longitud de la carga (payload): Indica el número de bytes de payload, es decir, indica la longitud de payload encapsulada en un registro. Se encuentra dentro de los primeros 8 bytes de un registro. El campo PAYLOAD_LENGTH es un byte para registros pequeños y cuatro bytes para registros normales. Los registros pequeños están indicados estableciendo el bit del flag SR (Short Record) en 1.
- Tipo de Payload: Indica la clase de datos que está siendo transportado en el payload de ese registro. El tipo del primer registro, por convención, debería proveer el contexto de procesamiento no solo para el primer registro sino para todo el mensaje NDEF. Los tipos de identificadores podrían ser URIs, MIME o tipos específicos NFC (NFC-specific). Al identificar el tipo de carga útil, es posible despachar la carga para la aplicación del usuario apropiada.
- Identificador de Payload: El payload puede dar un identificador opcional en la forma de una URI absoluta o relativa; esto permite a las cargas que soportan URI vincular tecnologías de referencia con otras cargas.

NDEF es simplemente un formato de mensaje, es decir que solo especifica la estructura del formato por lo que no se debe pensar que declara algún tipo de circuito o algún concepto de conexión o que pueda especificar el intercambio de información. El formato de datos de NDEF es el mismo tanto para un dispositivo NFC como para una etiqueta NFC, por lo que la información de NDEF es independiente del tipo de dispositivos que se estén comunicando.

Dentro del formato de un mensaje NDEF se puede enviar un variado tipo de información como:

- Puede encapsular documentos XML, fragmentos XML, datos encriptados, e imágenes como archivos JPEG, GIF, etc.
- Encapsular cadenas de información.
- Agregar documentos múltiples y entidades que están asociados lógicamente de alguna manera. Por ejemplo, se puede encapsular un mensaje NFC-specific y un

conjunto de archivos adjuntos de tipos estandarizados que tienen referencia desde ese mensaje NFC-specific.

- Encapsulado compacto de pequeños payloads.

Formato del Registro NDEF

Los registros NDEF son de longitud variable pero todos tienen un formato común que se representa a continuación con la siguiente figura:

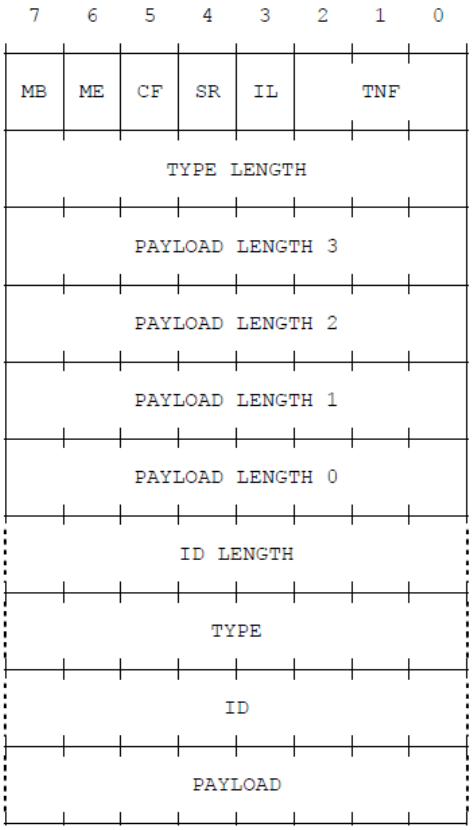


Ilustración 8: Formato mensaje NDEF[8]

La información de los registros NDEF se presenta a nivel de byte. El orden de transmisión es de izquierda a derecha y de arriba hacia abajo; de esta manera el bit más significativo del byte es el bit del extremo izquierdo y para una cadena de bytes es igual, el bit más significativo es el del lado izquierdo de todo el campo de bytes y es el que se transmite primero.

A continuación se detallan los campos que conforman el formato del registro NDEF:

- MB (Message Begin): Es un flag de 1 bit que cuando se constituye indica el inicio de un mensaje NDEF.

[8] Imagen obtenida de:[20] NFC Data Exchange Format (NDEF) NFC Forum, NDEF 1.0,NFCForum-TS-NDEF_1.0 2006-07-24

- ME (Message End): Este flag es un campo de 1 bit que si se establece, ya que en el caso de un payload fragmentado esta bandera solo se establece en el segmento de terminación de este payload fragmentado, indica el final de un mensaje NDEF.
- CF (Chunk Flag): Es un flag de 1 bit que de establecerse indica que es el primer segmento de registro o que es un segmento de registro del medio de un payload fragmentado.
- SR (Short Record): Se conforma por 1 bit y al establecerse indica que el campo PAYLOAD_LENGTH es un solo byte y no cuatro como lo es para un registro NDEF normal. Este registro pequeño está destinado para una encapsulación compacta la cual permite que pequeños payloads sean parte de campos de payloads con un tamaño de entre 0 a 255 octetos. Un mismo mensaje NDEF podría tener tanto registros NDEF normales como registros cortos.
- IL (ID_LENGTH): El flag IL es de 1 bit que si se establece indica que el campo ID_LENGTH está presente en la cabecera del registro como un byte pero si el campo IL es cero entonces éste es omitido de la cabecera y el campo ID también es omitido del registro.
- TNF (TYPE NAME FORMAT): Es un campo de 3 bits que indica la estructura del valor del campo TYPE. Estos valores se detallan a continuación:

Type Name Format	Valor
Vacío	0x00
Tipo NFC Forum (NFC RTD)	0x01
Tipo de Medios	0x02
URI Absoluto	0x03
Tipo NFC Forum externo	0x04
Tipo Desconocido	0x05
Sin Cambio (Unchanged)	0x06
Reservado	0x07

Tabla 1: Tabla de valores TNF

EL valor 0x00 (vacío) significa que no hay ningún tipo o payload asociada al registro. De esta manera los campos TYPE_LENGTH, ID_LENGTH y PAYLOAD_LENGTH deben ser cero y por lo tanto los campos TYPE, ID y PAYLOAD respectivamente serían omitidos del registro.

El valor 0x05 (Unknown) debería ser usado para indicar que el campo de payload es desconocido. Este valor de TNF ocasiona que el campo TYPE sea omitido del registro NDEF ya que el valor del campo TYPE_LENGTH debe ser cero. Se recomienda que cuando un analizador NDEF esté recibiendo un registro NDEF de este tipo, provea un mecanismo para guardar pero no para procesar la payload.

EL valor 0x06 (Unchanged) no debe usarse en otro registro que no sean los fragmentos de registro del medio y el fragmento del registro terminal que forman payloads segmentados. Si se establece este valor TNF, el campo TYPE_LENGTH debe ser cero y el campo TYPE será omitido del registro NDEF.

El valor 0x07 (Reservado) es para usos futuros y no debe ser usado.

- TYPE_LENGTH: Este campo es un entero no asignado de 8 bits que representa la longitud en bytes del campo TYPE. Al referirse a un entero no asignado, quiere decir que no es una constante sino que su valor depende de la longitud del campo TYPE.
- ID_LENGTH: Este campo también es un entero no asignado de 8 bits que especifica la longitud del campo ID en bytes y está presente sólo si el flag IL en la cabecera del registro se establece en 1.
- PAYLOAD_LENGTH: Es un entero no asignado que representa la longitud en bytes del campo PAYLOAD y a su vez el tamaño del campo PAYLOAD_LENGTH depende del valor del flag SR. Si el flag SR está establecido, el campo PAYLOAD_LENGTH representa un solo byte; pero si este flag está vacío, el campo PAYLOAD_LENGTH es de 4 bytes representando un entero no asignado de 32 bits.
- TYPE: Este campo es un identificador que especifica el tipo de payload de la información transmitida. El valor de este campo debe seguir la codificación, la estructura y el formato implícito por el valor del campo TNF. El tamaño máximo de este campo es 255 octetos.
- ID: El valor de este campo es un identificador que tiene la forma de una referencia URI (Identificador de Recursos Uniformes). Para NDEF, una URI es simplemente una cadena de texto que identifica un nombre, una localización o alguna característica de un determinado recurso. La singularidad requerida del identificador del mensaje es garantizada por el generador. Los fragmentos finales y de la mitad de un payload segmentado no debe tener el campo ID ya que se trata del mismo campo de datos pero en diferentes fragmentos por lo que solamente basta con definir una vez la información completa acerca de todo el payload. Todos los demás tipos de registros podrían tener este campo ID. El tamaño máximo de este campo es 255 bytes.
- PAYLOAD: Dentro de este campo se lleva la carga o información útil para las aplicaciones del usuario y la estructura interna de los datos llevados en este campo es oculta para NDEF. El tamaño máximo del campo PAYLOAD es 2^32-1 bytes para un diseño de registro NDEF normal y 255 bytes para un registro pequeño. Pero para tamaños de payload mayores a 2^32-1 se segmenta dicho payload para poder ser transmitido en fragmentos.

Mensaje NDEF

Un mensaje NDEF está compuesto por uno o varios Registros NDEF. El primer registro de un mensaje está marcado con la bandera MB (Message Begin) y el último registro lleva la bandera ME (Message End). Si un mensaje está compuesto por un solo registro, éste mismo lleva tanto la bandera MB como la bandera ME. El número de registros que un mensaje NDEF puede llevar es ilimitado.

Los mensajes NDEF no deben superponerse, es decir, que las banderas MB y ME no deben ser utilizadas para anidar mensajes NDEF. Los mensajes NDEF pueden ser anidados llevando un mensaje completo como una payload en un registro NDEF.

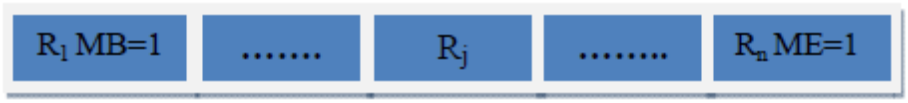


Ilustración 9: Registros en mensajes NDEF

Fragmentos de Registros

En realidad los mensajes NDEF no llevan números índices para indicar su orden, sino que está implícito en el orden en que los registros son almacenados. Por ejemplo si los registros son empaquetados por una aplicación intermedia, ésta es la responsable de asegurar que el orden de los registros sea el mismo.

Un registro es la unidad para llevar un payload en un mensaje NDEF y cada payload es descrita por sus propios parámetros.

Un pedazo de registro (record chunk) lleva solo un pedazo de payload. Estas cargas en pedazos son utilizadas para particionar un contenido generado dinámicamente o mensajes demasiados largos en múltiples pedazos de registro ordenados en un mismo mensaje NDEF. Este agrupamiento sirve para reducir la necesidad de almacenamiento en el búfer de salida en el lado generador.

Un mensaje NDEF puede contener cero o más payloads fragmentados. Cada payload fragmentado es codificado con las siguientes reglas:

- El pedazo de registro inicial tiene la bandera CF (Chunk Flag). El tipo de todo el payload fragmentado debe estar indicado en el campo de tipo. El campo ID podría ser usado para llevar un identificador de toda el payload fragmentado. El campo de la longitud de payload indica solo el tamaño de los datos que lleva el registro actual y no el tamaño de todo el payload.
- El fragmento de registro medio tiene la bandera CF indicando que este registro contiene el próximo pedazo de datos del mismo tipo y con el mismo identificador como el fragmento de registro inicial. El valor del campo TYPE_LENGTH y del campo ID_LENGTH debe ser cero y el campo TNF (Type Name Format) debe ser 0x06.

- El pedazo de registro final es un registro NDEF con la bandera CF limpia indicando que este fragmento de registro es el que contiene el último fragmento de datos del mismo tipo y con el mismo identificador que el fragmento del registro inicial. El valor del campo TYPE_LENGTH y del campo ID_LENGTH debe ser cero y el campo TNF (Type Name Format) debe ser 0x06.

Un payload fragmentado debe ser enteramente encapsulado en un sólo mensaje NDEF, o sea que no debe abarcar múltiples mensajes NDEF. Por lo tanto, ni un fragmento de registro inicial o medio puede tener una bandera ME establecida.

RTD, Definición de Tipo de Registro

La especificación RTD, por sus siglas en inglés Record Type Definition, provee las pautas para la especificación de los tipos de registros NFC bien conocidos (well-known record types) que puedan ser incluidos en mensajes NDEF transmitidos entre dispositivos NFC o entre un dispositivo NFC y una etiqueta NFC. Es decir que esta especificación permite soportar aplicaciones específicas NFC.

En el campo de formato de los tipos de registros de un registro NDEF están contenidos los nombres del tipo de registro llamado “record type name”. Cada definición de tipo de registro es identificado por su Record Type Name. Los Record Type Name pueden ser especificados en distintos formatos llamados Type Name Format (TNF) como ya se vio anteriormente. Estos podrían ser URIs absolutos, tipos NFC bien conocidos, tipos de medios MIME, tipos externos NFC.

Tipo NFC well-known

El tipo bien conocido NFC (NFC well-known) es un formato diseñado para las etiquetas NFC y también para crear formatos primitivos, es decir un formato primitivo, es decir un formato común, por ejemplo, se puede usar este tipo cuando no hay un equivalente URI o no hay disponible un tipo MIME o también cuando las limitaciones de un mensaje requieren un nombre muy pequeño.

Cuando un mensaje NDEF lleva un tipo NFC well-known en sí, el campo TNF tiene el valor 0x01.Los ejemplos más comunes son:

- Texto RTD [22]: La Definición de Tipo de Registro de Texto (Texto RTD) fue diseñado con el objetivo de definir un registro NFC well-known que contenga solamente datos de texto y que sea liviano para ser usados sin que se requiera de mucho espacio. El nombre que se utiliza para representar este tipo de registro es “T”.

Offset	Content	Explanation	Syntactical info
0	N/A	IL flag = 0 (no ID field), SF=1 (Short format)	NDEF record header
1	0x01	Length of the record name	
2	0x10	The length of the payload data (16 bytes)	
3	"T"	The binary encoding f the name	
4	0x02	Status byte: This is TF-8, and has a two-byte language code	Payload
5	"en"	"en" is the ISO code for "English"	
7	"Hello, world!"	UTF-8 string "Hello, world!"	The actual body text

Tabla 2: Ejemplo de registro NDEF tipo texto[9]

- URI RTD [23]: La Definición de Tipo de Registro URI no permite acceder a recursos de internet o para transportar los identificadores de Recursos, URI, de un dispositivo a otro. Es básicamente un contenedor compacto de URI's. El nombre de este registro para poder identificarlo es "U".

Offset	Content	Explanation
0	0xD1	SR = 1, TNF = 0x01 (NFC Forum Well Known Type), MB=1, ME=1
1	0x01	Length of the Record Type (1 byte)
2	0x0D	Length of the payload (13 bytes)
3	0x55	The Record Name ("U")
4	0x05	Abbreviation for "tel:"
5	0x2b 0x33 0x35 0x38 0x39 0x31 0x32 0x33 0x34 0x35 0x36 0x37	The string "+35891234567" in UTF-8.

Tabla 3: Ejemplo de registro NDEF tipo URI[10]

- Smart Poster RTD [21]: La Definición de Tipo de Registro Smart Poster especifica la manera de incorporar datos como SMS, URI's o números de teléfono en etiquetas NFC o la manera de transportarlos entre dispositivos. El objetivo de los Smart Poster es proveer una manera simple para acceder a un servicio remoto

[9] Tabla de:[22] Text Record Type Definition
NFC Forum, RTD-Text 1.0, NFCForum-TS-RTD_Text_1.0 2006-07-24

[10] Tabla de: [23] URI Record Type Definition
URI Record Type Definition

por un toque. Un Smart Poster también puede contener acciones que desplieguen una aplicación en un dispositivo, por ejemplo iniciar un explorador de internet. El nombre que identifica el registro Smart Poster es "Sp". Dentro de la payload de un Smart Poster hay algunos tipos de registros y pueden haber uno o más de estos registros (Text,URI,..)

Offset	Content	Length	Explanation
0	0xD1	1	NDEF header TNF = 0x01(Well Known Type). SR=1,MB=1, ME=1
1	0x02	1	Record name length (2 bytes)
2	0x49	1	Length of the Smart Poster data (73 bytes)
3	"Sp"	2	The record name
5	0x81	1	NDEF header. TNF = 0x01, SR=0, MB=1, ME=0
6	0x01	1	Record name length (1 byte)
7	0x00,0x00,0x00,0x0E	4	The length of the URI payload (14 bytes) (long format)
11	"U"	1	The length of the URI payload (14 bytes) (long format)
12	0x01	1	Abbreviation: "http://www."
13	"nfc-forum.org"	13	The URI itself.
26	0x11	1	NDEF record header (SR=1, TNF=0x01)
27	0x03	1	The length of the record name
28	0x01	1	The length of the "act" payload.
29	"act"	3	Record type: "act"
32	0x00	1	Action = Launch browser
33	0x11	1	NDEF record header (SR=1, TNF=0x01)
34	0x01	1	Length of the record name
35	0x12	1	Length of the record payload (18 bytes)
36	"T"	1	Record type: "T" (=Text)
37	0x05	1	Status byte for the Text (UTF-8, five-byte code)
38	"en-US"	5	ISO Language code: USEnglish
43	"Hello, world"	12	The text: "Hello world", encoded in UTF-8.
55	0x51	1	NDEF record header (SR=1, TNF= 0x01, ME=1)
56	0x01	1	Record name length
57	0x13	1	Length of the Text payload (19 bytes)
58	"T"	1	The name of the Text record ("T")

59	0x02	1	Status byte: UTF-8, two-byte language code
60	"fi"	2	ISO two-character language code: Finnish
62	"Morjens, maailma"	16	The text "Morjens, maailma" encoded in UTF-8

Tabla 4: Ejemplo de registro NDEF tipo Smartposter[11]

5.1.7 Estándares de NFC

El funcionamiento del NFC queda definido en 2 estándares, que se resumen en NFCIP-1 y NFCIP-2, aunque también hace uso de otros para facilitar la compatibilidad.

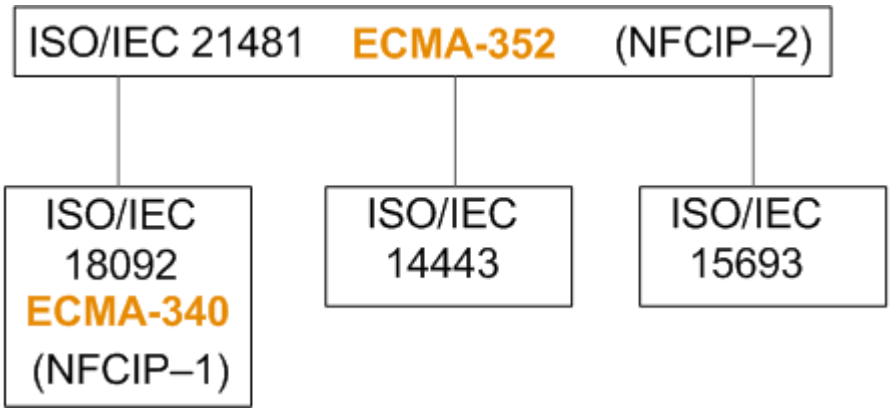


Ilustración 10: Estándares en NFC

ISO/IEC 18092 / ECMA-340 (NFCIP-1)

Esta norma específica los esquemas de modulación, codificación, velocidad de transferencia, y el formato de la trama de la interfaz de RF, así como esquemas de inicialización y las condiciones requeridas para el control de colisión de datos durante la inicialización.

ISO/IEC 14443

Este estándar defina las bases para trabajar con tarjetas de proximidad, usadas especialmente para la identificación.

ISO/IEC 15693

Es un estándar para tarjetas de vecindad (Vicinity Cards) que pueden ser leídas a una distancia mayor que las tarjetas de proximidad. Opera en la misma banda de frecuencia.

[11] Tabla de:[23] URI Record Type Definition
URI Record Type Definition

ISO/IEC 21481 / ECMA-352

Los 3 estándares anteriores trabajan en la misma banda de frecuencia (13.56 MHz), sin embargo tienen diferentes modos de comunicación. Estos modos de comunicación de definen como NFC, PCD, PICC, y VCD, en el orden visto anteriormente. NFCIP-2 especifica los mecanismos para detectar y seleccionar el modo de comunicación de esos 4 posibles casos.

ISO / IEC 18000

Es el estándar de RFID. Si bien este estándar tiene muchos puntos en común con NFC (trabaja el misma frecuencia, usa ISO 144443 e ISO 15693), son diferentes, ya que RFID puede trabajar a más distancia y en más bandas de frecuencia.

ISO / IEC 7816 e ISO / IEC 7810

Estándar para trabajar con tarjetas inteligentes independientemente del medio físico utilizado. En este se define el formato de las APDUs.

5.2 J2ME

Java 2 Micro Edition (J2ME) es un subconjunto de la plataforma Java diseñada para funcionar en "sistemas embebidos" como pueden ser reproductores DVD, electrodomésticos y en nuestro caso en dispositivos móviles.

5.2.1 Una introducción a JAVA

Java es un lenguaje de programación (y una plataforma informática) que se creó en 1991 cuando un grupo de ingenieros de Sun MicroSystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos.

Debido a la existencia de múltiples CPUs y los continuos cambios era necesario que crear una herramienta independiente de la CPU utiliza. Por esto mismo desarrollaron un compilador a lo que llamamos byte-code que es ejecutado sobre una máquina virtual, la conocida Java Virtual Machine (JVM). Esta máquina se encarga de convertir el byte-code al código particular de la CPU y sistema operativo utilizado. Esto permitió lo que posteriormente se convirtió en el lema del lenguaje "*Write once, run everywhere*" (WORA) que salvo excepciones particulares, las aplicaciones con un mismo código se pueden ejecutar en diferentes dispositivos, proporcionando un lenguaje independiente de la plataforma.

Las características principales de Java son:

- Orientación a objetos: Este es paradigma de programación utilizada en Java. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. Los objetos pueden heredar de otros objetos sus funciones y datos.
- Independencia de la plataforma: El código fuente java es compilado a byte-code, el cual contiene instrucciones específicas para la JVM. En cada plataforma existirá una JVM que se encargara de ejecutar dicho byte-code en esa máquina específica.
- Recolector de Basura: La recolección de basura simplifica la tarea de programar ya que el propio entorno de ejecución Java se encarga de liberar memoria eliminando los objetos que dejen de ser utilizados.
- Diferentes tipos de aplicaciones: Aplicaciones autónomas, applets, servlets, aplicaciones con interfaz gráfica, etc.

En la siguiente tabla se pueden ver las diferencias entre las distintas versiones de Java:

	J2EE	J2SE	J2ME
Dispositivo	Servidor	PC sobremesa	Móvil
API	Extendida	Amplia	Reducida
Memoria	MB	MB	KB

Tabla 5: Versiones de Java

5.2.2 Configuraciones y perfiles[12]

Las configuraciones se componen de una máquina virtual y un conjunto mínimo de bibliotecas. Proporcionan la funcionalidad básica para un conjunto de dispositivos que comparten características similares, tales como gestión de memoria o conectividad a la red. Existen 2 configuraciones definidas en J2ME: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.

5.2.2.1 CDC

La CDC está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Ésta Máquina Virtual es la CVM

[12] Texto extraído del documento:
Java a Tope: J2ME
Autores: Sergio Gálvez Rojas, Lucas Ortega Díaz

(Compact Virtual Machine). La CDC está enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits.
- Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM.
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

La CDC está basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones http y basadas en datagramas.

5.2.2.2 Perfiles

L os perfiles proporcionan al diseñador de los productos flexibilidad para soportar distintos tipos de dispositivos móviles personales con entornos de aplicaciones Java compatibles. Dado que las necesidades de las familias de dispositivos son muy diferentes, dentro de J2ME se han definido distintos perfiles para proporcionar opciones en función de las necesidades existentes:

- Foundation Profile (JSR-46): Es el perfil CDC básico. Junto con la biblioteca de clases que proporciona CDC se obtienen las clases de soporte básico de acceso a red y E/S. No proporcionan clases para gráficos o IGU
- Personal Basis Profile (JSR-129): Proporciona la estructura para crear componentes ligeros y soporte para el modelo de programación de aplicaciones Xlet. Incluye las APIs del Foundation Profile
- Personal Profile (JSR-62): Proporciona soporte completo para AWT, applet y soporte limitado para beans. Incluye todas las APIs del Personal Basis Profile.

5.2.2.3 CLDC

Esta configuración está diseñada para dispositivos con conexiones de red intermitentes, procesadores lentos y memoria limitada como son teléfonos móviles, asistentes personales (PDAs), etc. Está orientado a dispositivos que cumplan las siguientes características:

- Procesador: 16 bit/16 MHz o más.
- Memoria: 160-512 KB de memoria total disponible para la plataforma Java.
- Alimentación limitada, a menudo basada en batería.
- Trabajo en red: Conectividad a algún tipo de red, con ancho de banda limitado habitualmente.

Podemos clasificar los requerimientos en hardware, software y requerimientos basados en las características Java de la plataforma J2ME.

Requerimientos hardware: CLDC está diseñado para ejecutarse en una gran variedad de dispositivos, desde aparatos de comunicación inalámbricos como teléfonos móviles, buscapersonas, hasta organizadores personales, terminales de venta, etc. Las capacidades del hardware de estos dispositivos varían considerablemente y por esta razón, los únicos requerimientos que impone la configuración CLDC son los de memoria. La configuración CLDC asume que la máquina virtual, librerías de configuración, librerías de perfil y la aplicación debe ocupar una memoria entre 160 Kb y 512 Kb. Más concretamente:

- 128 Kb de memoria no volátil para la JVM y las librerías CLDC.
- Al menos 32 Kb de memoria volátil para el entorno de ejecución Java y objetos en memoria. Este rango de memoria varía considerablemente dependiendo del dispositivo al que hagamos referencia.

Requerimientos software: Al igual que las capacidades hardware, el software incluido en los dispositivos CLDC varía considerablemente. Por ejemplo, algunos dispositivos pueden poseer un sistema operativo (S.O.) completo que soporta múltiples procesos ejecutándose a la vez y con sistema de archivos jerárquico. Otros muchos dispositivos pueden poseer un software muy limitado sin noción alguna de lo que es un sistema de ficheros. Dentro de esta variedad, CLDC define unas mínimas características que deben poseer el software de los dispositivos CLDC. Generalmente, la configuración CLDC asume que el dispositivo contiene un mínimo Sistema Operativo encargado del manejo del hardware de éste. Este S.O. debe proporcionar al menos una entidad de planificación para ejecutar la JVM. El S.O. no necesita soportar espacios de memoria separados o procesos, ni debe garantizar la planificación de procesos en tiempo real o comportamiento latente.

Requerimientos J2ME: CLDC es definida como la configuración de J2ME. Esto tiene importantes implicaciones para la especificación CLDC:

Una configuración J2ME solo deber definir un complemento mínimo de la tecnología Java. Todas las características incluidas en una configuración deben ser generalmente aplicables a una gran variedad de dispositivos. Características específicas para un dispositivo, mercado o industria deben ser definidas en un perfil en vez de en el CLDC. Esto significa que el alcance de CLDC está limitado y debe ser complementado generalmente por perfiles.

El objetivo de la configuración es garantizar portabilidad e interoperabilidad entre varios tipos de dispositivos con recursos limitados. Una configuración no debe definir ninguna característica opcional. Esta limitación tiene un impacto significativo en lo que se puede incluir en una configuración y lo que no. La funcionalidad más específica debe ser definida en perfiles.

5.2.2.4 MIDP

Este perfil está construido sobre la configuración CLDC. Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma. Este perfil está orientado para dispositivos con las siguientes características:

- Reducida capacidad computacional y de memoria.
- Conectividad limitada (en torno a 9600 bps).
- Capacidad gráfica muy reducida (mínimo un display de 96x54 pixels monocromo).
- Entrada de datos alfanumérica reducida.
- 128 Kb de memoria no volátil para componentes MIDP.
- 8 Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

Los tipos de dispositivos que se adaptan a estas características son: teléfonos móviles, buscapersonas o PDAs de gama baja con conectividad. El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las APIs relacionadas con:

- La aplicación (semántica y control de la aplicación MIDP).
- Interfaz de usuario.
- Almacenamiento persistente.
- Trabajo en red.
- Temporizadores.

En la siguiente imagen se puede ver el cido de ejecución de un midlet, las aplicaciones desarrolladas para J2ME:

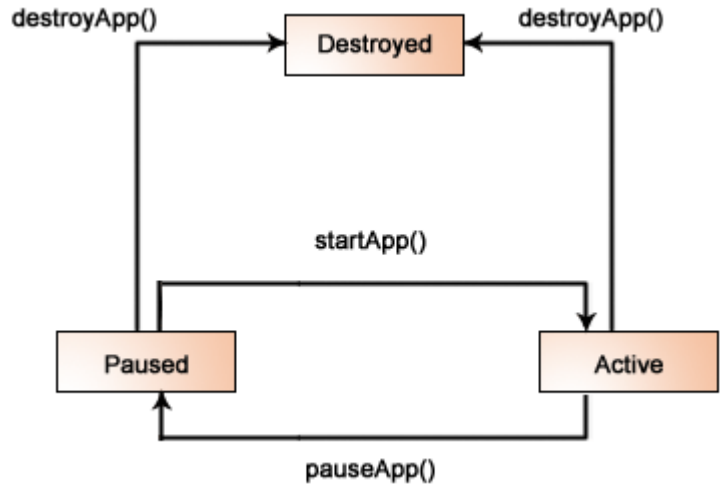


Ilustración 11: Ciclo de ejecucción de un Midlet

Cuando se inicia la aplicación, el primer método al que se llama es al método constructor. Inmediatamente después se llama al método startApp() y la aplicación

comienza a estaren primer plano. Cuando deja de estar activa y pasa a segundo plano, se ejecuta el método pauseApp(), donde se realizan las tareas necesarias para reducir los recursos que estan utilizados. En el momento que la aplicación vuelve a estar activa, se vuelve a llamar al método starApp(). Cuando se finaliza definitivamente la aplicación, se llama al método destroyApp() desde cualquiera de los 2 estados.

5.2.3 API´s y NFC

El JCP[1] (Java Community Process, Proceso de la Comunidad Java) es el organismo que gestiona las diferentes versiones de java, permitiendo a sus representantes involucrarse en el desarrollo y revisiones de las especificaciones de Java. Actualmente participan un gran número de empresas: IBM, Oracle, Nokia, Ericsson, HP, Gemalto, Google y más.

Las diferentes especificaciones se aprueban en JSR (Java Specification Request) en el cual se describen las tecnologías y requisitos propuestos que se van añadir a la plataforma Java. Estas se convierten en finales una vez han sido aprobadas por el Comité Ejecutivo JCP. Un JSR final ofrece:

- Especificación en formato electrónico (PDF).Es un documento que describe la tecnología utilizada y las especificaciones que se den de cumplir. Se pueden descargar públicamente.
- Test de compatibilidad (TCK): Asegura la compatibilidad de las diferentes implementaciones por cada fabricante. Aunque no es necesario pasar estos test de calidad, sirven como garantía de calidad. Éstas si es obligatorio que pasen el test de compatibilidad.
- Implementación de referencia (RI): Son implementaciones sencillas que sirven para probar las especificaciones incluso antes de su lanzamiento. Estas si es obligatorio que pasen el test de compatibilidad.

El propio JCP queda descrito en un JSR (JSR 215).

Ciclo de desarrollo:

- Envío de la especificación(1 mes).El JCP la aprobará o denegara
- Revisión por parte de la comunidad (2-3 meses).Los miembros del JCP opinan y se realiza una votación para aceptarla
- Revisión pública (3-6 meses) Todo el mundo puede participar. En esta fase se termina el desarrollo del TCK y RI. Si todo va bien se publica la versión final.
- Fase de mantenimiento. Correcciones y mejoras.

En el caso de J2ME es interesante que los teléfonos móviles en función del hardware y el soporte que se haya facilitado en su KVM implementaran unas JSR o no. Por

ejemplo, la JSR-82 corresponde con las especificaciones de las comunicaciones Bluetooth, por lo que solo los teléfonos móviles con este tipo de hardware pueden implantarlo. Además para hacer uso de las librerías descritas en la JSR, el fabricante debe haberlo implementado en la KVM del dispositivo.

Algunos ejemplos de JSR que están disponibles el Nokia 6212

JSR 177: Security and Trust Services

Servicios de seguridad y confianza. Esas API proporcionan mecanismos de seguridad para soportar una amplia variedad de servicios de aplicaciones basados, como el acceso a la red corporativa, comercio móvil, y la gestión de derechos digitales. Muchos de estos servicios se basan en la interacción con un elemento de seguridad (Secure Element) en el dispositivo para el almacenamiento y ejecución segura. Tipos de aplicaciones:

- Almacenamiento seguro para proteger información sensible: claves privadas del usuario, certificados de clave pública, credenciales de servicio e información personal.
- Ejecución segura, operaciones criptográficas para poder soportar protocolos de pago, integridad y confidencialidad de los datos.
- Características de seguridad que permite que las aplicaciones J2ME sean fiables para manejar servicios de valor añadido, tales como la identificación y autenticación de usuarios, operaciones bancarias, pagos, emisión de billetes, aplicaciones de la lealtad, la reproducción de contenido multimedia digital, etc.

Los principales paquetes son `javax.microedition.se` y `javax.microedition.crypto`

JSR 75: File Connection and PIM File system, contacts, calendar, to-do

Esta especificación se divide en dos paquetes independientes:

- Gestión de la información personal (PIM): Este paquete permite acceder a los datos personales de forma nativa. Información entre la que se encuentran la libreta de direcciones, el calendario y listas de tareas pendientes.
- Acceso a ficheros(FC).Permite el acceso a los sistemas de ficheros presentes en el teléfono móvil. En principal uso es la posibilidad de acceder a los dispositivos de almacenamiento extraíble, como las tarjetas de memoria externa.

Los nombres de los principales paquetes son `javax.microedition.io` y `javax.microedition.pim`.

JSR 82: Bluetooth

Se trata de un conjunto de API que permite que los dispositivos habilitados se integren en un entorno Bluetooth. Incluye soporte básico para al menos protocolos RFCOMM, OBEX y descubrimiento de servicios. El primero proporciona conexiones básicas a los dipositivos. El segundo proporciona la capacidad de intercambio de información orientado a objetos, y es la base para los perfiles de nivel superior como puede ser la sincronización de la agenda telefónica o el intercambio de ficheros. Se define de tal manera que pueda extenderse a futuros protocolos de comunicación sobre Bluetooth. Además puede incluirse en entornos que no sean estrictamente J2ME (Por ejemplo BlueCove para la versión J2SE) .Nombre del paquete principal `javax.microedition.bluetooth`.

JSR 257: Contactless Communication API (NFC)

Esta es la API más importante para el proyecto ya que permite multitud de funciones: Lectura/escritura de un numero so tipo de tarjetas, conexión NFCIP. Está especialmente desarrollada para dispositivos con recursos limitados (J2ME). El nombre del paquete es `javax.microedition.contactless`.

5.3 Android

Es un SO basado en Linux diseñado para funcionar en dispositivos móviles como smartphones o *tablets*. Fue desarrollado por Android Inc. Y posteriormente comprado por Google quien siguió con su desarrollo. La principal ventaja de Android es el un SO común para los diferentes fabricantes de móviles, pudiendo estos personalizar Android a su manera ya que se distribuye bajo licencia Apache. Salvo algún problema puntual de compatibilidad, todas las aplicaciones pueden correr en los diferentes modelos siempre y cuando tengan la misma versión o superior.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núdeo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución. El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2,8 millones de líneas de lenguaje C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

5.3.1 Características y arquitectura

Las principales características de Android son:

- Es in SO libre distribuido bajo licencia Apache: Con esta licencia se encuentra disponible el código fuente para su uso libre, y cada fabricante puede modificarlo el SO a su antojo, sin tener que estar obligado a publicar el código fuente con los cambios (a diferencia de lo que ocurriría con una licencia GPL). Además ha permitido que se crearan diferentes comunidades que han modificado Android con sus preferencias y modificaciones particulares, como por ejemplo CyanogenMod o MIUI.

- Está desarrollado en Java y C. Para programar en esta plataforma se usa principalmente Java. Para casos excepcionales (tareas de rendimiento) es posible programarlo en C gracias al SDK Bionic. Actualmente hay iniciativas para el uso de nuevos lenguajes(c#,..) pero el recomendado es java.

- Soporte para un gran número de comunicaciones: 3G, Wi-Fi, Bluetooth, Wimax, NFC y más.

- Navegador web basado en Webkit: Otros navegadores como Chrome o Safari están basados también en este, lo que facilita la compatibilidad y futuras correcciones de vulnerabilidades y errores,

- Soporte para Flash: Aunque no es recomendable en dispositivos móviles, debido entre otras cosas al gasto de batería que supone, para una navegación web completa a día de hoy sigue siendo importante la posibilidad de instalar flash en Android, hasta que termine de implantarse HTML5.

- Soporte para compartir la red (tethering): Es posible compartir la conexión de datos contratada con un PC u otros móviles a través del USB o creando una red Wi-Fi.

- Amplio soporte multimedia.

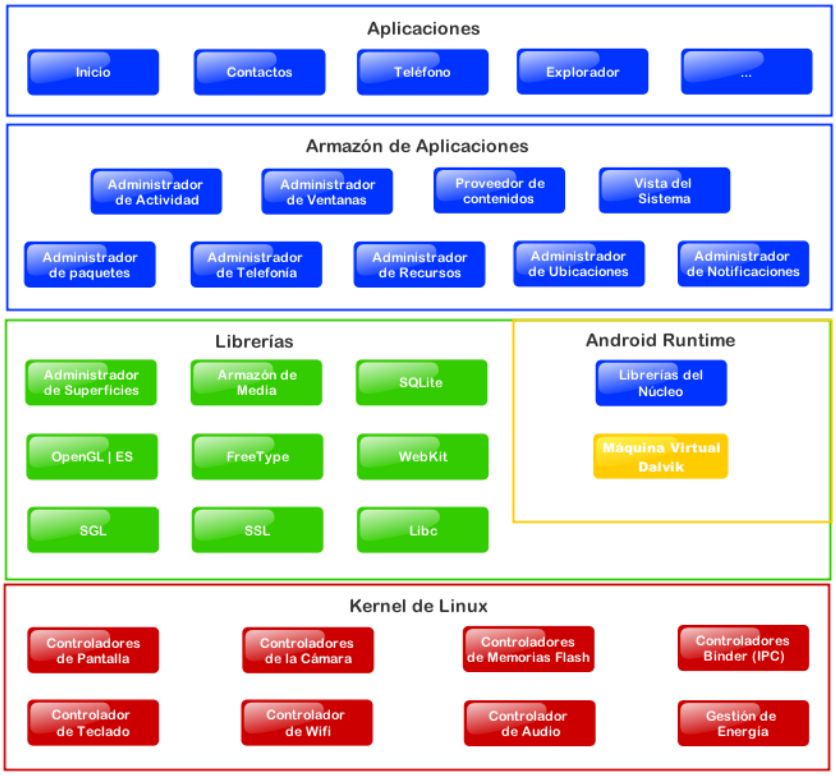


Ilustración 12 Arquitectura Android[13]

- Kernel de Linux: Utiliza una versión 2.6.x. Actúa como capa de abstracción entre el hardware y el resto de la pila de software. Hace uso de los servicios base del sistema como es la seguridad, la gestión de memoria, la gestión de proceso, la pila de la red y el modelo de drivers.
- Librerías Android: Con junto de librerías en C/C++ usadas por varios componentes del sistema. Se expone a los desarrolladores a través del framework de Android, y entre otra son: System C library (implementación biblioteca C estándar), Surface Manager (Gestión de acceso a la pantalla),Media Framework (Reproducción de contenido multimedia) SQlite (pequeña base de datos), WebKit (motor de renderizado web), SGL (gráficos 2D) Open GL ES (gráficos 3D).
- Android Runtime: La máquina virtual Dalvick para ejecutar las aplicaciones escritas en java. Cada aplicación Android tiene su propia instancia de la máquina virtual. Está optimizado para trabajar con este tipo de dispositivos: Limitación por la batería, memoria y procesamiento. Los ejecutables están en formato dex (Dalvik Executable).
- Framework (Marco de trabajo) de Android: Conjunto de API utilizadas por las aplicaciones, que permiten a los desarrolladores acceder a toda la funcionalidad y todas las características de Android y del dispositivo. La

[13] Imagen obtenida de: http://es.wikipedia.org/wiki/Android

arquitectura está planificada para simplificar la reutilización de componentes: cualquier aplicación puede publicar sus propias funciones para que otras aplicaciones hagan uso de estas. A través del framework el desarrollador puede acceder a la información de ubicación, estado de la red, información del teléfono y de la SIM, cámara y el resto de dispositivos de hardware, a alto nivel.

- Aplicaciones: En esta última capa se encuentra las aplicaciones de las que hará uso el usuario, tanto preinstaladas como las desarrolladas y posteriormente instaladas. Correo electrónico, agenda, calendario, gestor de mensajes (SMS), Android Market.

5.3.2 SDK y versiones (2.2,2.3,2.3.3)[14]

Desde su aparición en 2008 han salido numerosas actualizaciones. Éstas corrigen fallos, implementan nuevas funcionalidades, mejoran el rendimiento, etc. Las aplicaciones desarrollas son por regla general retrocompatibles.

Los correspondientes SDK salen poco tiempo después de cada actualización.

1.5 (Cupcake)

Basado en el kernel de Linux 2.6.27 El 30 de abril de 2009.Características nuevas y actualizaciones en la interfaz de usuario en la actualización 1.5:

- Posibilidad de grabar y reproducir videos a través del modo camcorder.
- Capacidad de subir videos a YouTube e imágenes a Picasa directamente desde el teléfono.
- Un nuevo teclado con predicción de texto.
- Soporte para Bluetooth A2DP y AVRCP.
- Capacidad de conexión automática para conectar a auricular Bluetooth a cierta distancia.
- Nuevos widgets y carpetas que se pueden colocar en las pantallas de inicio.
- Transiciones de pantalla animadas.

1.6 (Donut)

Basado en el kernel de Linux 2.6.29 El 15 de septiembre de 2009:

- Una experiencia mejorada en el Android Market.
- Una interfaz integrada de cámara, filmadora y galería.
- La galería ahora permite a los usuarios seleccionar varias fotos para eliminarlas.

[14] Características extraídas de: http://es.wikipedia.org/wiki/Android

- Búsqueda por voz actualizada, con respuesta más rápida y mayor integración con aplicaciones nativas, incluyendo la posibilidad de marcar a contactos.
- Experiencia de búsqueda mejorada que permite buscar marcadores, historiales, contactos y páginas web desde la pantalla de inicio.
- Actualización de soporte para CDMA/EVDO, 802.1x, VPN y text-to-speech.
- Soporte para resoluciones de pantalla WVGA.
- Mejoras de velocidad en las aplicaciones de búsqueda y cámara.
- Framework de gestos y herramienta de desarrollo GestureBuilder.
- Navegación gratuita turn-by-turn de Google.

2.0 / 2.1 (Eclair)

Basado en el kernel de Linux 2.6.29 El 26 de octubre de 2009:

- Velocidad de hardware optimizada.
- Soporte para más tamaños de pantalla y resoluciones.
- Interfaz de usuario renovada.
- Nuevo interfaz de usuario en el navegador y soporte para HTML5.
- Nuevas listas de contactos.
- Una mejor relación de contraste para los fondos.
- Mejoras en Google Maps 3.1.2.
- Soporte para Microsoft Exchange.
- Soporte integrado de flash para la cámara.
- Zoom digital.
- MotionEvent mejorado para captura de eventos multi-touch.
- Teclado virtual mejorado.
- Bluetooth 2.1.
- Fondos de pantalla animados.

2.2 (Froyo)

Basado en el kernel de Linux 2.6.32 El 20 de mayo de 2010:

- Optimización general del sistema Android, la memoria y el rendimiento.
- Mejoras en la velocidad de las aplicaciones, gracias a la implementación de JIT.
- Integración del motor JavaScript V8 del Google Chrome en la aplicación Browser.
- Soporte mejorado de Microsoft Exchange (reglas de seguridad, reconocimiento automático, GAL look-up, sincronización de calendario, limpieza remota).
- Lanzador de aplicaciones mejorado con accesos directos a las aplicaciones de teléfono y Browser.
- Funcionalidad de Wi-Fi hotspot y thetering por USB.
- Permite desactivar el tráfico de datos a través de la red del operador.
- Actualización del Market con actualizaciones automáticas.

- Cambio rápido entre múltiples idiomas de teclado y sus diccionarios.
- Marcación por voz y compartir contactos por Bluetooth.
- Soporte para contraseñas numéricas y alfanuméricas.
- Soporte para campos de carga de archivos en la aplicación Browser.
- Soporte para la instalación de aplicación en la memoria expandible.
- Soporte para Adobe Flash 10.1.
- Soporte para pantallas de alto número de Puntos por pulgada, tales como 4". 720p.

2.3 (Gingerbread)

Basado en el kernel de Linux 2.6.35.7 El 6 de diciembre de 2010:

- Actualización del diseño de la interfaz de usuario.
- Soporte para pantallas extra grandes y resoluciones WXGA y mayores.
- Soporte nativo para telefonía VoIP SIP.
- Soporte para reproducción de videos WebM/VP8 y decodificación de audio AAC.
- Nuevos efectos de audio como reverberación, ecualización, virtualización de los auriculares y refuerzo de graves.
- Soporte para Near Field Communication.
- Funcionalidades de cortar, copiar y pegar disponibles a lo largo del sistema.
- Teclado multi-táctil rediseñado.
- Soporte mejorado para desarrollo de código nativo.
- Mejoras en la entrada de datos, audio y gráficos para desarrolladores de juegos.
- Recolección de elementos concurrentes para un mayor rendimiento.
- Soporte nativo para más sensores (como giroscopios y barómetros).
- Un administrador de descargas para descargar archivos grandes.
- Administración de la energía mejorada y control de aplicaciones mediante la administrador de tareas.
- Soporte nativo para múltiples cámaras.
- Cambio de sistema de archivos de YAFFS a ext4.

3.0 / 3.1 (Honeycomb)

- Mejor soporte para tablets.
- Escritorio 3D con widgets rediseñados.
- Sistema multitarea mejorado.
- Mejoras en el navegador web predeterminado, entre lo que destaca la navegación por pestañas, autorelleno de formularios, sincronización de favoritos con Google Chrome y navegación privada.
- Soporte para videochat mediante Google Talk.

4.0 (Ice Cream Sandwich)

- Interfaz estilo Honeycomb, en cualquier dispositivo, homogeneidad entre teléfonos, televisiones, tablets, netbooks.
- Barra de estado redimensionable.
- Reconocimiento de voz del usuario.
- Reconocimiento facial, lo que haría que puedas cambiar la vista.
- Un único y nuevo framework para las aplicaciones.

5.3.3 NFC en Android

Con la versión 2.3 comienza el soporte de NFC sobre Android. Pero la API facilitada a los desarrolladores no permite aprovechar todas las capacidades que NFC debería ofrecer, por lo que en poco espacio de tiempo Google lanzó una versión mejorada. Esta nueva versión mejoro el soporte de tarjetas y añadió la posibilidad de enviar mensajes NDEF entre teléfonos.

Las posibilidades ofrece Android para trabajar con NFC son las siguientes:

- Lectura escritura de diferentes tipos de tarjeta: Todas las posibles tarjetas están representadas por la clase Tag. Las propiedades de esta, permiten obtener el tipo de tarjeta aproximada y en función de la que sea, obtener la clase que la representa ese tipo. En el paquete `android.nfc.tech` se encuentran las clases disponibles, que ofrecen soporte para los siguientes tipos de tarjeta: NFC-A (ISO 14443-3A), NFC-B (ISO 14443-3B), NFC-F (JIS 6319-4), NFC-V (ISO 15693), ISO-DEP (ISO 14443-4), MIFARE Classic, MIFARE Ultralight y NFC Forum NDEF tags.
- Formateado de diferentes mensajes NDEF: Permite trabajar alto nivel con los diferentes tipos de mensajes NDEF (texto, URL,..) tanto en la lectura como la escritura.
- Posibilidad de intercambiar mensajes NDEF entre teléfonos móviles: A modo de conexión peer to peer, se pueden enviar o recibir mensajes NDEF como si de una tarjeta se tratase.
- Aviso de aproximación de un elemento NFC: Cuando se aproxima una tarjeta u otro móvil, se puede preparar para que la aplicación que este en primer plano reciba el correspondiente aviso, con un *Intent* (como suele trabajarse en Android, a modo de "Evento") o se puede configurar para elegir que aplicación se abrirá por defecto en caso de no estar ejecutándose ninguna aplicación relacionada.

6 Aplicación de inicio de sesión

La aplicación de divide en varias partes que se fueron desarrollando cronológicamente. En primer lugar se desarrolló una API para poder trabajar con el lector. El propósito de esta API es que sea lo suficientemente funcional y completa para que pueda usarse para todo tipo de aplicaciones. En las primeras versiones, fue necesario desarrollar una parte en C para poder comunicarse con el lector. En versiones posteriores no fue necesario gracias a la utilización de una librería propia de java para trabajar con lectores de tarjetas inteligentes. Veremos las diferencias de una y otra posteriormente.

El siguiente paso necesario, fue la implementación de las funcionalidades NFC con sus protocolos, desde comunicaciones NFCIP, emulación de tarjetas y lectura y escritura de diferentes tipos de tarjetas.

Una vez conseguida la base, se procedió a la puesta en práctica de la idea un ejemplo de a autentificación. Se desarrolló un protocolo de actuación para la aplicación. Se desarrolló la aplicación, diente, servidor y la configuración en el servidor (servicio, BBDD).

Las partes en las que se divide funcionalmente la aplicación, podrían permitir el funcionamiento de diferentes maneras, teniendo estos 3 escenarios como principales modelos.

Tabla 6: Modos de funcionamiento

Home y base de datos local: Con esta configuración, la aplicación puede funcionar en un solo ordenador de forma autónoma. Tanto los datos de usuario como la base de

datos con las claves de acceso se encuentran el propio ordenador donde se encuentra el lector y el servicio de login NFC ejecutándose.

Home local y base de datos remota: En esta configuración, los datos del usuario están en la propia máquina, mientras que la base de datos se encuentra en una máquina remota. De esta manera se tiene la ventaja de una mejor gestión de las claves de acceso, que en caso de robo o pérdida se puede anular o cambiar muy fácilmente. Además permite por ejemplo el montaje de una red en las distintas aulas con un usuario por defecto común para todos, que no guarde datos personales de ningún tipo.

Home y base de datos remota: Esta tercera configuración sería idéntica a la anterior con la ventaja de tener una cuenta con sus respectivos datos personales para cada usuario, en una red de ordenadores. Esta es la que más ventajas ofrece en los diferentes casos, como las aulas de ordenadores para los alumnos de una Universidad.

A pesar de que para el presente proyecto se realiza una aplicación muy concreta, hay que destacar que es la base para realizar cualquier tipo de aplicaciones.

Por un lado, la librería creada permite llevar a cabo gran parte de los usos del NFC. Puede realizar comunicaciones con móviles, con etiquetas, tarjetas Mifare y se podría desarrollar una API para trabajar con tarjetas inteligentes. Todo esto permite los diferentes usos que le queramos dar al NFC: Tarjetas de información contacto, pagos, cuponing, ticketing, identificación, tarjetas con información (de ayuda en museos, aeropuertos,…) o publicidad (centros comerciales).

Por otro lado la aplicación NFC de la que hace uso el servicio, sirve como base para cualquier aplicación relacionada con la identificación como puede ser el control de acceso o servicios de check-in. Esto es gracias como está estructurado: Por un lado la aplicación de loginNFC se limita a mostrar el usuario que ha sido identificado. Por otro lado el servicio que utiliza el usuario mostrado por loginNFC para realizar el acceso. De esta manera podríamos construir otra aplicación o servicio que hiciera uso de la primera parte, para poder utilizarla en otros casos completamente diferentes: control de acceso a un edificio, confirmación de llegada a una cita al ambulatorio u hospital, uso de cajeros, etc.

6.1 Comenzando a trabajar con el lector

La comunicación con el lector se puede hacer de 2 formas: Usando el API de java (javax.smarcardio) o usando un API en C (winscard).En los primeros comienzos se hizo uso de winscard para poder realizar comunicaciones NFCIP, pero finalmente se decidió utilizar el API de java por su sencillez y fácil implementación. De todas formas se habla brevemente del modo de funcionamiento inicial que quedo obsoleto cuando se decidió realizar emulación de tarjetas.

6.1.1 Winscard

Winscard es una librería que se puede utilizar tanto en Windows como en GNU/Linux, con cualquier dispositivo PC/SC. Dispone de una gran cantidad de métodos de los cuales nos interesan 4:

- ScardEstablishContext
- ScardListReaders
- SCardConnect
- ScardControl

Los 3 primeros son para conectarse al lector, el último es para enviar recibir información. Por eso a la hora de hacer uso desde java lo simplificamos en 2 procedimientos: iniciar y lectorctl. Con estos 2 procedimientos cubriremos todas nuestras necesidades.

Para llamar a estos métodos tendremos que incluir los siguientes headers*: reader.h* y *winscard.h*.

6.1.2 JNI y librerías dinámicas DLL/SO

A través de la JNI se hace una conversión entre los métodos que necesariamente hemos tenido que desarrollar en C y sus llamadas desde java. Para ello es necesario declararlos como nativos (native), posteriormente prepara el archivo header pasándole el archivo javah. Una vez tenemos el header procederemos a implementar los métodos.

Java dispone de diferentes tipos de datos: int, double, byte,…Estos mismos tipos también están en C y a para pasar de unos a otros existe una equivalencia directa.

Para nuestro caso tenemos necesitaremos pasar 2 tipos de datos boolean y byte[].En C estos 2 tipos están definidos en el header jni.h como jboolean y jByteArray. En el caso de los valores boolean podemos tener true o false con JNI_FALSE o JNI_TRUE. Para jByteArray es un poco más complicado: el método GetByteArrayElements pasaremos el array de Java a C y con SetByteArrayRegion pasaremos un puntero de C a un array de Java.

Esta vía de desarrollo se dejó de utilizar porque finalmente no fue necesario, pero aun así hay una parte desarrollada y documentada por si fuera de utilidad en futuras aplicaciones.

6.1.3 El paquete javax.smartcardio

Es el paquete en java (definido en la JSR 268) que describe una API para realizar comunicaciones con tarjetas inteligentes. Los resultados que se pueden obtener son los mismos que con Winscard en C, utilizando el lector como un PCSC y enviando APDUs con las instrucciones.

La única diferencia es que para poder transmitir información debe estar presente algún elemento NFC en las proximidades del lector, si no es imposible realizar la comunicación. Debido a este detalle, en el caso de comunicaciones NFCIP si es necesario utilizar Winscard ya que el dispositivo móvil no es detectado por el lector. En el resto de los casos (lectura/escritura de tarjetas, emulación) es recomendable utilizar esta librería que nos ofrece java por su sencillez y su fácil portabilidad. Se puede ver en el siguiente ejemplo:

```
// Mostrar los lectores (PCSC) conectados al PC
TerminalFactory factory = TerminalFactory.getDefault();
List<CardTerminal> terminals = factory.terminals().list();
System.out.println("Terminals: " + terminals);
// Utilizar el primer lector de la lista
CardTerminal terminal = terminals.get(0);
// Conectar con la tarjeta disponible
Card card = terminal.connect("T=0");
System.out.println("card: " + card);
CardChannel channel = card.getBasicChannel();
ResponseAPDU r = channel.transmit(new CommandAPDU(c1));
System.out.println("response: " + toString(r.getBytes()));
// Desconectar
card.disconnect(false);
```

Las clases utilizadas se corresponden con los siguientes elementos:

- TerminalFactory: Para obtener los lectores disponibles.
- **CardTerminal**: El Lector propiamente.
- **Card:** La tarjeta o elemento NFC disponible.
- **CardChannel:** La vía de comunicación entre el lector y los elementos.
- CommandAPDU/ResponseAPDU: Bytes enviados y recibidos.

Posteriormente se explica con detalle el uso y funcionamiento

Diferencias entre la versión 1 y la versión 2

	V1	V2
Dependencias	Depende de DLL (win)/ SO(linux)	javax.samrtcardio.*
NFCIP	Nokia	--
ISODEP	Emulacion SCard14emu Funciona con Nokia y Nexus	Emulacion con ISO14emu Funciona con Nokia y Nexus
Etiquetas soportadas	Topaz	Mifare1k, MifareUltraligth, (Topaz también implementable)

Tabla 7: Diferencias entre versiones de la librería

En conclusión, se utiliza la versión 2 de la librería, por su sencillez, mayor portabilidad entre sistemas operativos y compatibilidad con las diferentes plataformas móviles utilizadas. En el siguiente capítulo se describe con detalle cómo se implementa

cada funcionalidad y como están organizados los paquetes y clases. Los resultados obtenidos nos ofrecen las siguientes ventajas:

- Multipropósito: Es perfectamente válido para leer tarjetas, operar con móviles.
- Información de depuración: Si bien la librearía permite interactuar con el lector y las comunicaciones NFC a alto nivel, también es mostrar por pantalla diferentes niveles de información de depuración para ayudar con la detección de errores o implementaciones de futuros usos.
- Modo directo e indirecto (eventos).

6.2 Modos de funcionamiento

En este punto se trata el funcionamiento del lector, independientemente del lenguaje de programación que se utilice para trabajar con él. El principal objetivo es encontrar una forma de comunicar el lector con el Nokia 6212 y el Nexus S, pero se explican todos los modos de funcionamiento: Leer/escribir tarjetas, emular smartcards y comunicaciones vía NFCIP.

Todos estos modos de funcionamiento se implementan mediante el intercambio de diferentes APDUs al lector. La estructura de las APDU queda definida en el estándar ISO/IEC 7816- Parte 4. Se utilizan con cualquier lector de tarjetas inteligentes (PCSC) y con cualquier tipo de tarjeta, siendo independiente del medio físico utilizado (por contacto o radiofrecuencia).Hay 2 tipos, las de envío y las de recepción (CommandAPDU y ResponseAPDU respectivamente)

CommandAPDU						
Cabecera(Fijo)				Contenido (Variable)		
CLA	INS	P1	P2	Lc	Datos	Le
1 byte	1 byte	1 byte	1 byte	0-3 bytes	Lc bytes	0-3 bytes

- CLA: Denota el tipo de comando, si es algún tipo propietario o no, así como la estructura y formato del CommandAPDU y ResponseAPDU
- INS: Indica el tipo de instrucción del comando (leer, escribir, obtener otro tipo de información,…).
- P1 y P2: Parámetros, dependen de la instrucción.
- Lc: Longitud en butes de los datos enviados.
- Le: Longitud en bytes de los datos que se esperan recibir.

ResponseAPDU		
Contenido (Varable)	Cola (Fijo)	
Datos	SW1	SW2
Le bytes	1 byte	1 byte

- SW(Status Word): Indican el estado de la operación. Los 2 más usados normalmente son 90 00 para denotar que la operación se realizó con éxito y 63 00 para indicar que hay algún problema o error.

Ejemplo para obtener identificador:

CLA	INS	P1	P2	Lc
FF	CA	00	00	00

Y se espera recibir algo como lo siguiente:

Datos	SW1	SW2
XX XX … XX	90	00

De ahora en adelante se explicara los posibles APDU que se deben utilizar para conseguir la funcionalidad requerida, pero sin entrar en detalle de su formato.

6.2.1 Mifare Ultraligth

Para comenzar, lo más sencillo es trabajar con una tarjeta de este tipo. Las acciones de lectura / escritura se limitan a enviar la instrucción de leer o escribir directamente sobre el bloque solicitado. Los procedimientos se realizan en grupos de 4 bytes para que coincida con el tamaño de bloque, aunque también sería posible leer de una vez hasta 16 bytes. Es importante recordar que con los 4 primeros bloques solo se pueden realizar operaciones de lectura (están reservados para el fabricante) y que el tamaño total es de 64 bytes.

```
FF B0 00 03 04
xx xx xx xx 90 00
```

Ejemplo de lectura del bloque 3.El segundo byte (INS=B0) representa la instrucción de lectura. El cuarto (P2=03) indica que el bloquea tratar es el tercero. El ultimo (Le=04) espera que la longitud de los datos recibidos será 4.

```
FF D6 00 05 04 01 02 03 04
90 00
```

Ejemplo de escritura de los bytes 01 02 03 04 en bloque 5.Ahora el segundo byte (INS=D6) contiene la instrucción de escritura. El cuarto byte(P2=05) igualmente indica el número de bloque como en el caso anterior y el quinto (Lc=4) el número de bytes que hay en los datos y que se van a escribir.

6.2.2 Tarjetas Mifare1K

La estructura de la memoria en las tarjetas Mifare 1K quedaba definida en 16 sectores con 3 bloques libres por sector 1 un bloque por sector para fijar los permisos de acceso.

Los pasos a seguir para trabajar con las tarjetas Mifare son los siguientes:

- Cargar la clave/claves en el lector.
- Autenticarse en el sector solicitado con la clave/claves cargadas
- Si la autenticación fue exitosa, podremos realizar cualquier operación de lectura/escritura (siempre y cuando no hayamos limitado los permisos anteriormente) con los mismo APDU que se utilizan con las Mifare Ultralight. Esto incluye la posibilidad de cambiar la clave o cambiar los permisos, escribiendo en el bloque correspondiente, por lo que debemos de tener cuidado.

```
FF 82 00 00 06 FF FF FF FF FF FF
90 00
```

Este APDU es el que se utiliza para preparar la clave que será utilizada, con el byte INS=82.Los datos es la clave (FF FF FF FF FF FF) y Lc=06 la longitud de ésta.

```
FF 86 00 00 05 01 00 00 60 00
90 00 / 63 00
```

Este es el APDU para realizar la autenticación en el bloque 5 con la clave previamente cargada en el lector. Si el ResponseAPDU es igual a 90 00, se puede proceder a la lectura o escritura (si se tienen permisos) de ese bloque.

```
FF B0 00 01 10
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx 90 00
```

La única diferencia respecto a las Mifare Ultralight en las APDU de lectura escritura es que siempre se hace en grupos de 16 bytes(Lc/Le=0x10) mientras en las otras esto era opcional.

6.2.3 Pseudo-APDUs

Los psuedo-apdus se utilizan para realizar transmisiones directas, lo cual es necesario en los siguientes casos:

- Intercambiar información con tarjetas no compatibles con PCSC.
- Obtener y configurar parámetros del lector.
- Emulación de tarjetas.
- Comunicaciones peer to peer,NFCIP.

El formato de las pseudo-APDUs es el siguiente (la respuesta no tiene un formato predefinido):

CLA	INS	P1	P2	Lc	Datos
FF	00	00	00	N bytes	Payload

El resultado final de la convivencia entre los diferentes tipos de tarjeta puede contemplarse en este diagrama:

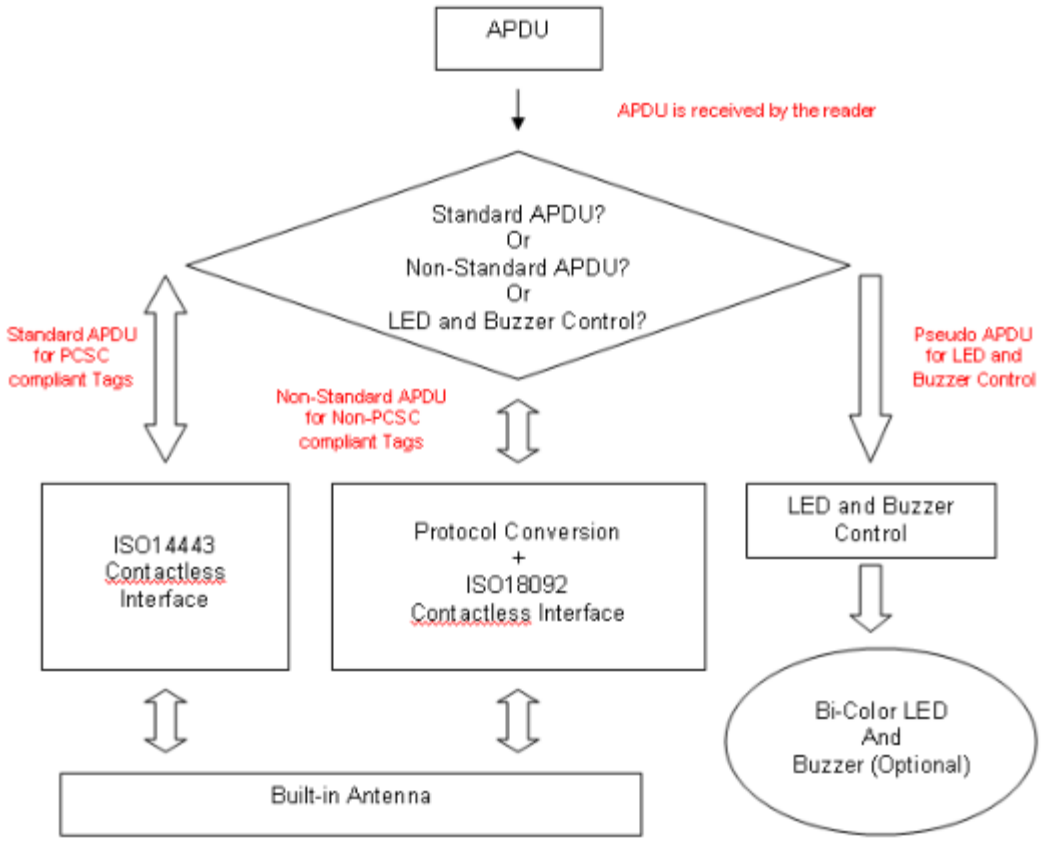


Ilustración 13: Funcionamiento Lector ACR122[15]

En función del APDU que le llegue al lector operará de un modo u otro. Como puede verse en la imagen, también se puede controlar el led del lector.

6.2.4 Emulacion de Smartcards

Este es uno de los modos de trabajo más interesantes. Permite comunicarse de forma transparente con cualquier lector o teléfono móvil que sea capaz de comunicarse con tarjetas inteligentes, con el estándar ISO 14443-4.

Esto además de facilitar la detección del lector, pues simplemente hay que configurar el dispositivo para que lo detecte, facilita la comunicación. Estos es debido a que en las APDUs que se enviarían a una tarjeta inteligente los datos pueden ir en cualquier formato, pudiendo realizar la comunicación de cualquier tipo de información.

En el lado del terminal, hay que tener en cuenta tan solo 3 comandos que por sencillez se nombraran como el datasheet del chip PN532[15]:

[15] Imagen obtenida de: [16] Funcionamiento del lector ACR122
ACR122 NFC Reader API

TgInitAsTarget: Configura el lector como *Target*. Con este comando el lector funcionara como una tarjeta ISO 14443-4A. Además se puede enviar el UID que identifica la tarjeta emulada.

TgGetData: Espera la recepción del primer conjunto de datos enviados desde el móvil a la tarjeta emulada en el lector

TgSetData: Responde a los datos anteriormente recibidos, realizando el envío desde la tarjeta emulada al móvil con el cual se está comunicando.

Para transmitir estos 3 comandos, será necesario utilizar transmisión directa:

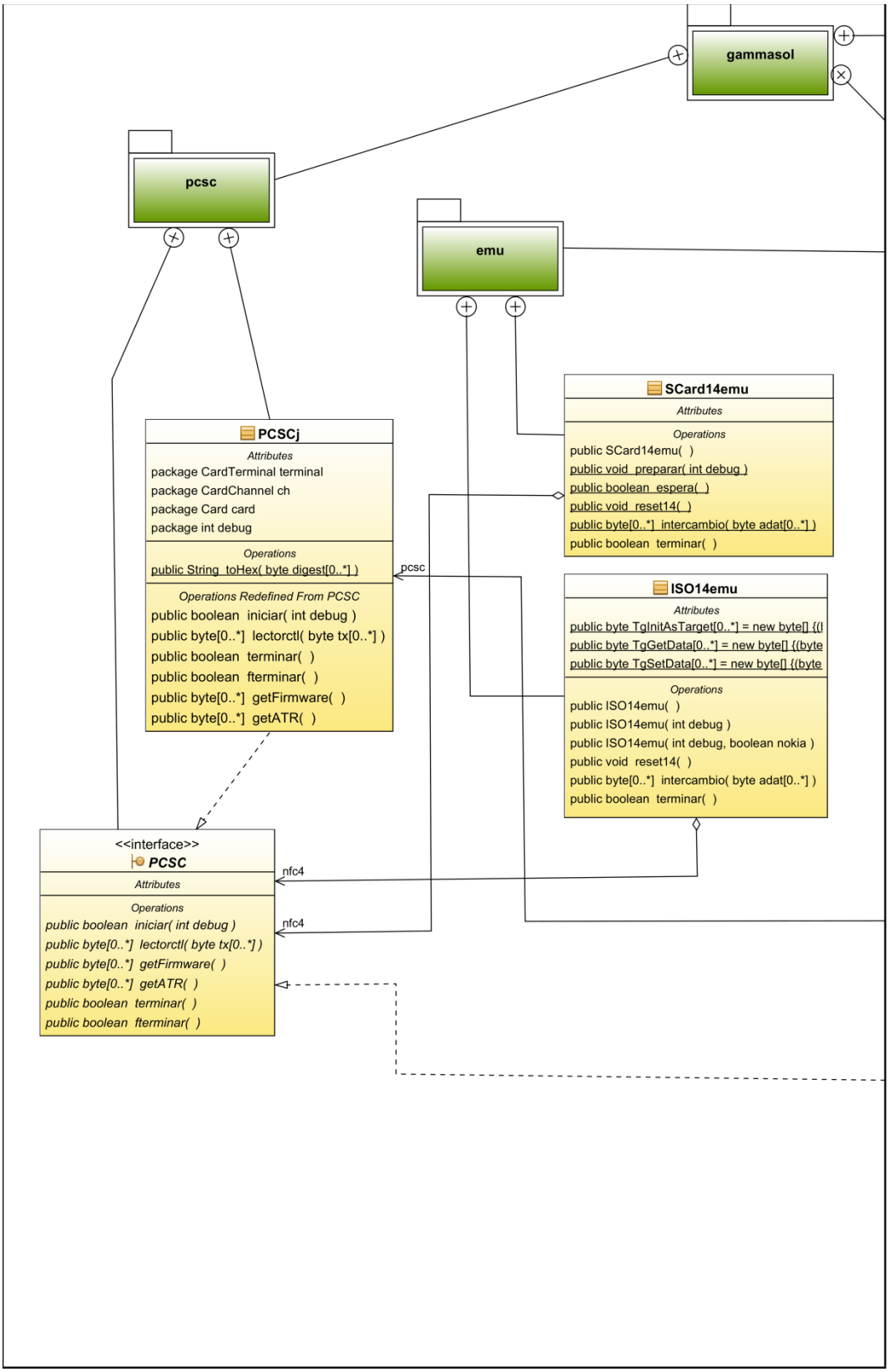
FF 00 00 00 Lc 'TgInitAsTarget'

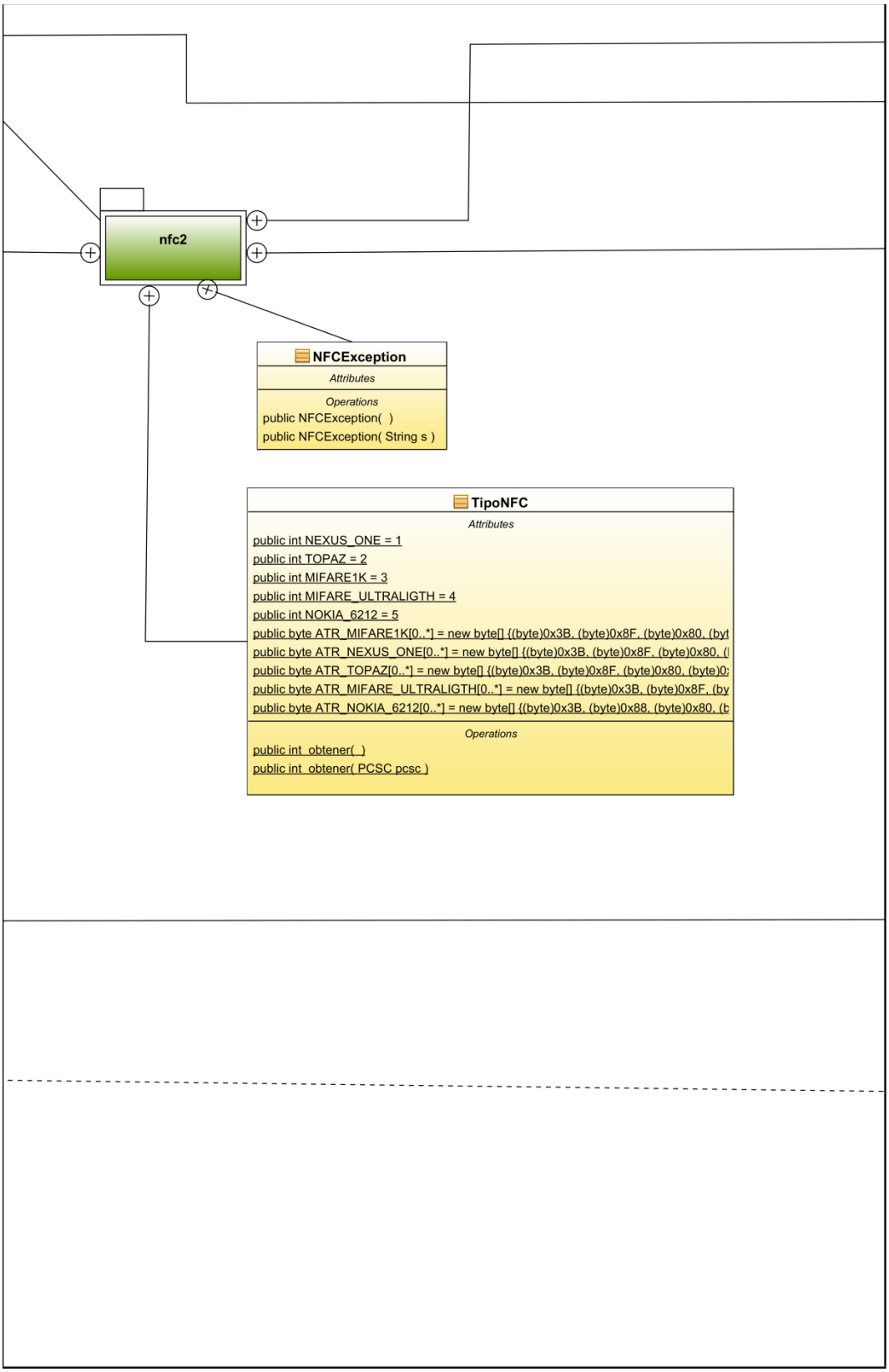
La complejidad del funcionamiento del chip pn532 es bastante elevada para entrar en detalle. Se puede obtener más información del documento del mismo, o del código implementado. De igual manera, se encuentran los pasos a seguir para realizar comunicaciones NFCIP.

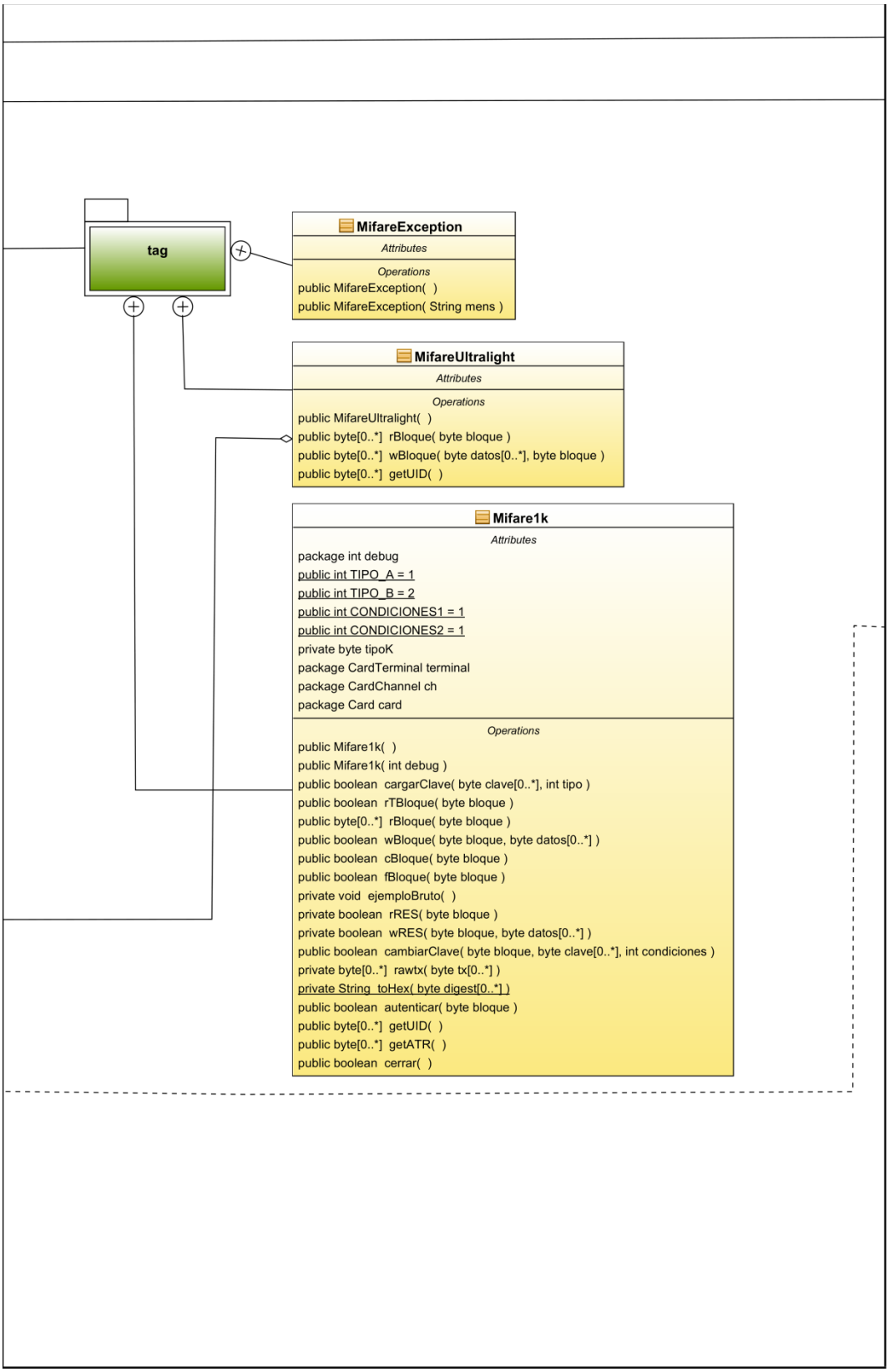
6.3 Implementación de la librería

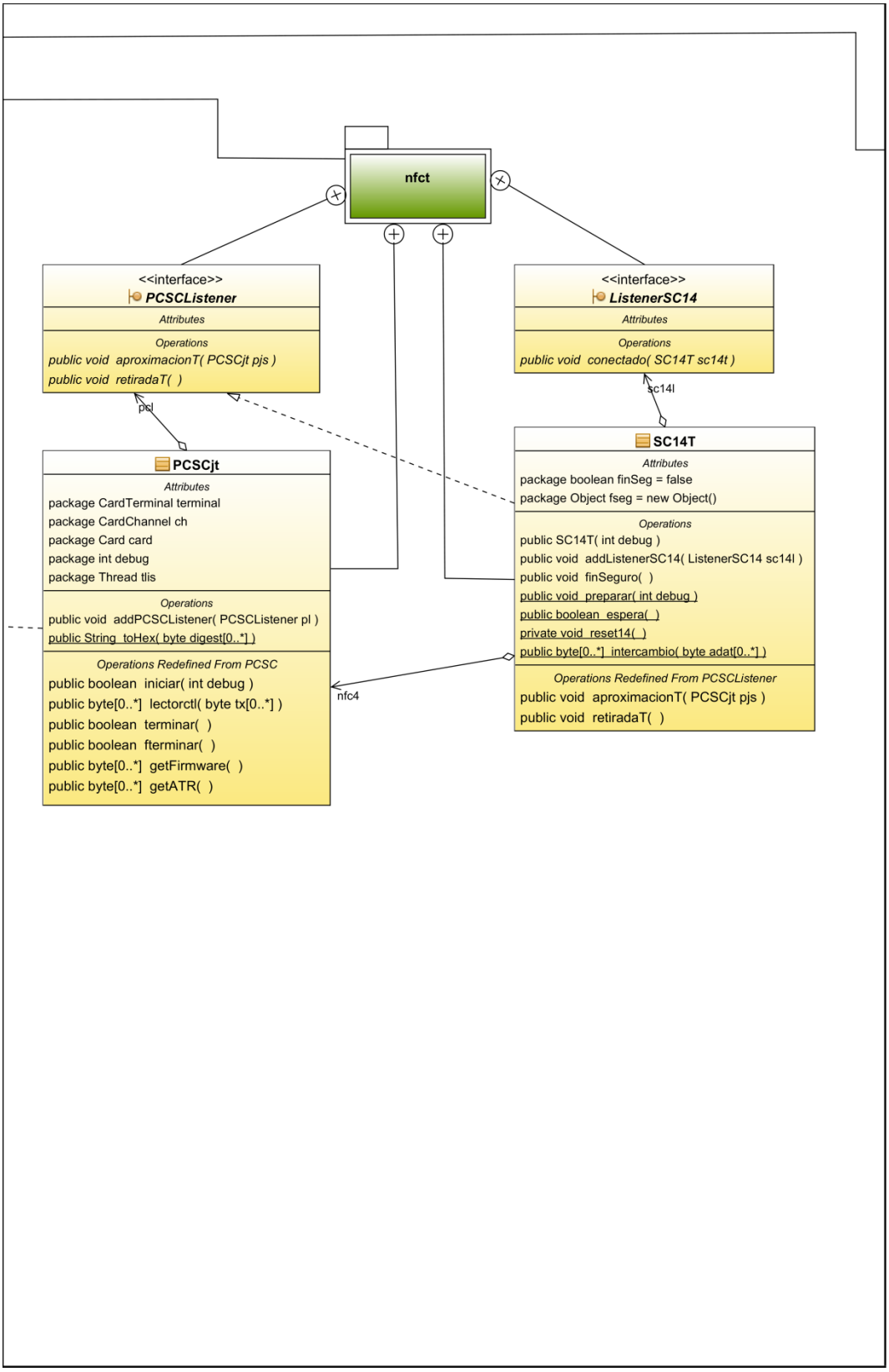
En este capítulo se detalla cómo funciona la librería, como está estructurada y como está escrita con detalle, de tal forma que pueda ser modificada de forma sencilla en caso de ser necesario incluir nuevas funcionalidades.

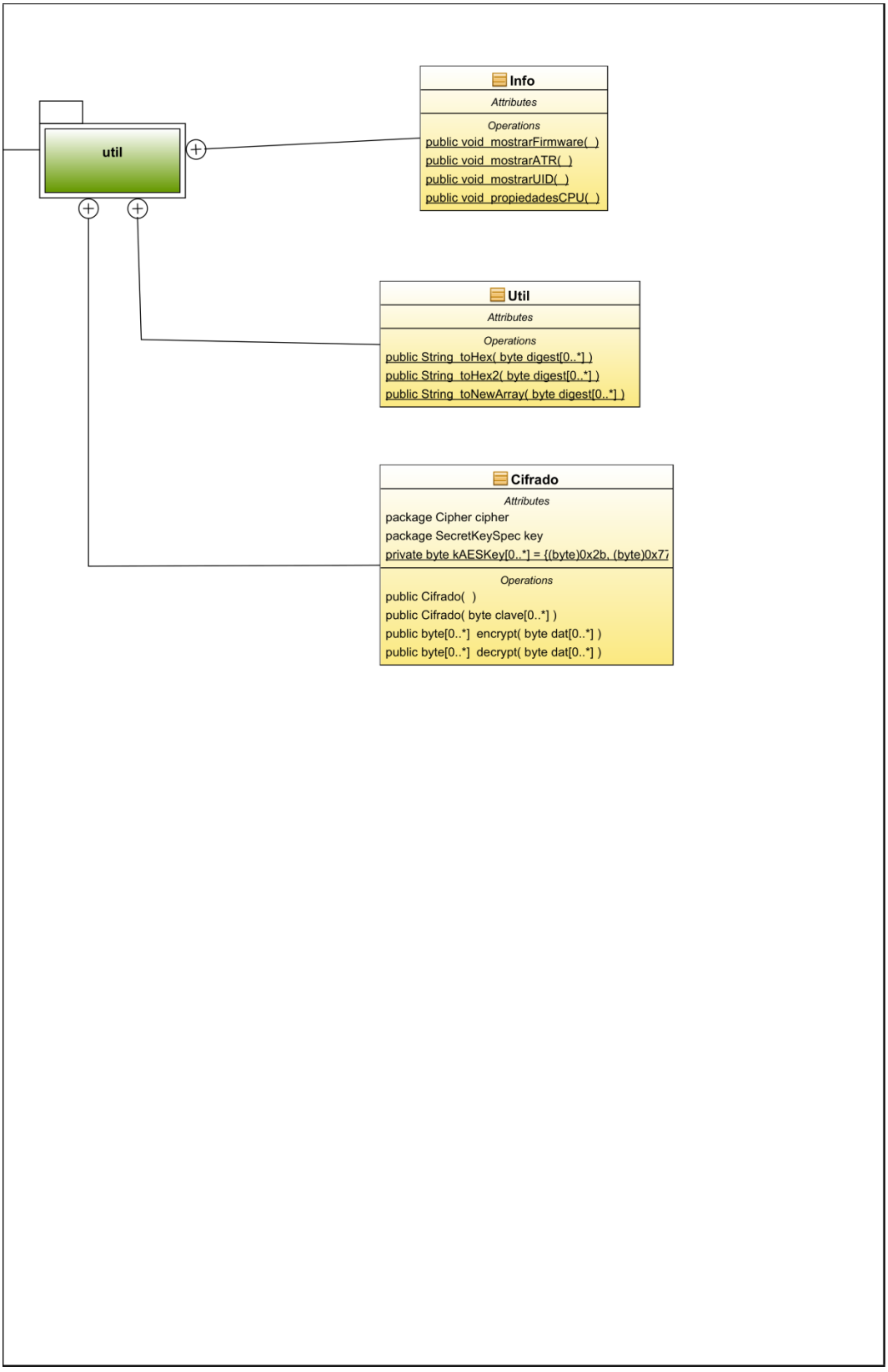
En primer lugar se muestra el diagrama UML correspondiente a todos los paquetes y clases, después se explica con detalle como esta implementada cada clase.


gammasol
pcsc
emu
SCard14emu
Attributes
Operations
public SCard14emu( )
public void preparar(int debug)
public boolean espera( )
public void reset14( )
public byte[0..*] intercambio(byte adat[0..*])
public boolean terminar( )
PCSCj
Attributes
package CardTerminal terminal
package CardChannel ch
package Card card
package int debug
Operations
public String toHex(byte digest[0..*])
Operations Redefined From PCSC
public boolean iniciar(int debug)
public byte[0..*] lectorctl(byte tx[0..*])
public boolean terminar( )
public boolean fterminar( )
public byte[0..*] getFirmware( )
public byte[0..*] getATR( )
pcsc
ISO14emu
Attributes
public byte TgInitAsTarget[0..*] = new byte[] {(
public byte TgGetData[0..*] = new byte[] {(byte
public byte TgSetData[0..*] = new byte[] {(byte
Operations
public ISO14emu( )
public ISO14emu(int debug)
public ISO14emu(int debug, boolean nokia)
public void reset14( )
public byte[0..*] intercambio(byte adat[0..*])
public boolean terminar( )
<<interface>>
PCSC
Attributes
Operations
public boolean iniciar(int debug)
public byte[0..*] lectorctl(byte tx[0..*])
public byte[0..*] getFirmware( )
public byte[0..*] getATR( )
public boolean terminar( )
public boolean fterminar( )
nfc4
nfc4


nfc2
NFCException
Attributes
Operations
public NFCException( )
public NFCException(String s)
TipoNFC
Attributes
public int NEXUS_ONE = 1
public int TOPAZ = 2
public int MIFARE1K = 3
public int MIFARE_ULTRALIGTH = 4
public int NOKIA_6212 = 5
public byte ATR_MIFARE1K[0..*] = new byte[] {(byte)0x3B, (byte)0x8F, (byte)0x80, (byt
public byte ATR_NEXUS_ONE[0..*] = new byte[] {(byte)0x3B, (byte)0x8F, (byte)0x80, (
public byte ATR_TOPAZ[0..*] = new byte[] {(byte)0x3B, (byte)0x8F, (byte)0x80, (byte)0:
public byte ATR_MIFARE_ULTRALIGTH[0..*] = new byte[] {(byte)0x3B, (byte)0x8F, (by
public byte ATR_NOKIA_6212[0..*] = new byte[] {(byte)0x3B, (byte)0x88, (byte)0x80, (b
Operations
public int obtener( )
public int obtener(PCSC pcsc)


tag
MifareException
Attributes
Operations
public MifareException( )
public MifareException(String mens)
MifareUltralight
Attributes
Operations
public MifareUltralight( )
public byte[0..*] rBloque(byte bloque)
public byte[0..*] wBloque(byte datos[0..*], byte bloque)
public byte[0..*] getUID( )
Mifare1k
Attributes
package int debug
public int TIPO_A = 1
public int TIPO_B = 2
public int CONDICIONES1 = 1
public int CONDICIONES2 = 1
private byte tipoK
package CardTerminal terminal
package CardChannel ch
package Card card
Operations
public Mifare1k( )
public Mifare1k(int debug)
public boolean cargarClave(byte clave[0..*], int tipo)
public boolean rTBloque(byte bloque)
public byte[0..*] rBloque(byte bloque)
public boolean wBloque(byte bloque, byte datos[0..*])
public boolean cBloque(byte bloque)
public boolean fBloque(byte bloque)
private void ejemploBruto( )
private boolean rRES(byte bloque)
private boolean wRES(byte bloque, byte datos[0..*])
public boolean cambiarClave(byte bloque, byte clave[0..*], int condiciones)
private byte[0..*] rawtx(byte tx[0..*])
private String toHex(byte digest[0..*])
public boolean autenticar(byte bloque)
public byte[0..*] getUID( )
public byte[0..*] getATR( )
public boolean cerrar( )


nfct
<<interface>>
PCSCListener
Attributes
Operations
public void aproximacionT(PCSCjt pjs)
public void retiradaT( )
<<interface>>
ListenerSC14
Attributes
Operations
public void conectado(SC14T sc14t)
pcsl
sc14l
PCSCjt
Attributes
package CardTerminal terminal
package CardChannel ch
package Card card
package int debug
package Thread tlis
Operations
public void addPCSCListener(PCSCListener pl)
public String toHex(byte digest[0..*])
Operations Redefined From PCSC
public boolean iniciar(int debug)
public byte[0..*] lectorctl(byte tx[0..*])
public boolean terminar( )
public boolean fterminar( )
public byte[0..*] getFirmware( )
public byte[0..*] getATR( )
SC14T
Attributes
package boolean finSeg = false
package Object fseg = new Object()
Operations
public SC14T(int debug)
public void addListenerSC14(ListenerSC14 sc14l)
public void finSeguro( )
public void preparar(int debug)
public boolean espera( )
private void reset14( )
public byte[0..*] intercambio(byte adat[0..*])
Operations Redefined From PCSCListener
public void aproximacionT(PCSCjt pjs)
public void retiradaT( )
nfc4


util
Info
Attributes
Operations
public void mostrarFirmware( )
public void mostrarATR( )
public void mostrarUID( )
public void propiedadesCPU( )
Util
Attributes
Operations
public String toHex(byte digest[0..*])
public String toHex2(byte digest[0..*])
public String toNewArray(byte digest[0..*])
Cifrado
Attributes
package Cipher cipher
package SecretKeySpec key
private byte kAESKey[0..*] = {(byte)0x2b, (byte)0x7
Operations
public Cifrado( )
public Cifrado(byte clave[0..*])
public byte[0..*] encrypt(byte dat[0..*])
public byte[0..*] decrypt(byte dat[0..*])

6.3.1 Organización de los paquetes

La estructura que se sigue es la siguiente:

- **gammasol.pcsc**: Contiene la interfaz PCSC y la implementación en java PCSCJ.
- **gammasol.nfc2**: Contiene la NFCException y TipoNFC.
- **gammasol.nfc2.tag**: Clases referentes a etiquetas y tarjetas (Mifare1k,Ultraligth).
- **gammasol.nfc2.emu**: Clases sobre la emulación con el lector (ISO14emu, NDEFemu en futuras versiones).
- **gammasol.nfc2.util**: Diferentes utilidades, como el cifrado, mostrar información del sistema, pasar bytes a String en hexadecimal…
- **gammasol.nfct**: Implementación de algunas clases con hilos, de forma similar a ”eventos y listeners”. EXPERIMENTAL.
- **ejemplos** : ejemplos de uso. Hay una clase Main que permite llamar al resto de los métodos *static* que podrían funcionar igualmente como Main.

6.3.2 PCSC

Esta se trata de la clase más importante de todas, ya que tiene los métodos básicos que permiten la comunicación entre el PC y cualquier lector (no solo NFC). Es la que se encarga de la espera de elementos NFC, detección, inicio de la conexión, envió y recepción de datos y cierre de la conexión. Esta clase es utilizada por todas las que hacen uso del lector.

La clase que implementa los métodos de esta interfaz es PCSCj, la cual esta implementada en java. Se deja la posibilidad de que en un futuro se pueda usar algún lector de diferente manera, pero que puedan implementarse los métodos de distinta forma o sea necesario hacer uso de la JNI.

Los métodos son:

- **iniciar(int debug):** Se queda a la espera de aproximar un elemento NFC e inicia la comunicación. Si debug=0 no mostrara información. Si debug=1, muestra algunos detalles. Con debug=3 mostrara los APDUs enviados y recibidos.
- **lectorctl(byte[] tx)**:Una vez iniciado, sirve para transmitir APDUs al lector y realizar el envío de datos.
- **getATR():** Obtiene el ATR (internamente se obtiene antes de la comunicación e identifica el tipo de elemento NFC)
- **terminar():** finaliza la comunicación de manera segura, esperando la retirada del elemento.
- **fterminar()** : finaliza la comunicación de manera brusca (forzada).

Pasamos a explicar con detalle como esta implementado cada método.

El método iniciar crea los objetos que representan al lector. En primer lugar con la clase TerminalFactory se pueden obtener los lectores conectados y disponibles. Como pueden existir varios conectados, se utiliza el primero de la lista por defecto

```
List<CardTerminal> terminals;
TerminalFactory factory = TerminalFactory.getDefault();
terminals = factory.terminals().list();
CardTerminal terminal = terminals.get(0);
```

Cuando está todo preparado, se entra en modo espera indefinidamente a que exista algún elemento NFC sobre el lector. Una vez se aproxime, se abre una vía de comunicación con la clase CardChannel. Card representa el propio elemento aproximado.

```
terminal.waitForCardPresent(0);
if (terminal.isCardPresent()) {
    card = terminal.connect("*");
    if (debug > 0) {
        System.out.println("Conectado");
    }
    ch = card.getBasicChannel();
        return true;
    }
```

Y de esta manera se ha iniciado la conexión. El método lectorctrl permite comunicarse con el lector directamente, pudiendo obtener información sobre la versión del firmware, leer tarjetas o emularlas para comunicarse con otros lectores o teléfonos móviles NFC.

```
CommandAPDU c = new CommandAPDU(tx);
ResponseAPDU r = ch.transmit(c);
rx = r.getBytes();
```

La clase CardChannel posee el método transmit, que servirá para enviar y recibir los APDUs. Los APDUs están representados por CommandAPDU para el envió y ResponseAPDU para la respuesta, pero no son más que un array de bytes con un formato determinado por la aplicación que se vaya a usar.

El método getATR obtiene el ATR del elemento detectado. El se obtiene previamente al inicio de la conexión, y representa la velocidad que soporta, el tipo de elemento que es, etc. No se debe confundir con el intercambio de APDUs, es un identificador previo a la conexión.Se obtiene con el método getATR() del objeto Card.

```
card.getATR().getBytes();
```

El método getFirmware realiza el envío de un APDU que específicamente con este lector, sirve para darle la instrucción de obtener la versión del firmware del lector. Si hay interacción entre el lector y el PC, pero no con el elemento que haya conectado.

```
lectorctl(new byte[]{(byte) 0xff, 0x00, 0x48, 0x00, 0x00});
```

Por último para finalizar la comunicación hay 2 maneras posibles: Comprobando que se ha retirado el elemento antes de finalizar o sin realizar esta comprobación. En el resto de los métodos hay comprobaciones del valor de debug, como se ve en el siguiente fragmento, para mostrar la información del funcionamiento por pantalla.

```
terminal.waitForCardAbsent(0);
if (debug > 0) {
    System.out.println("Finalizado");
}
card.disconnect(true);
```

6.3.3 TipoNFC

Sirve para saber previamente a la comunicación que tipo de elemento NFC se ha aproximado. Para ello, internamente lee el ATR y los comprueba con los almacenados, devolviendo el valor el tipo de elemento.

- **TipoNFC.obtener():**Devuelve el tipo de elemento TipoNFC.NEXUS_ONE , TipoNFC.MIFARE1k,TipoNFC.MifareUltraligth, etc.
- **TipoNFC.obtener(PCSC pcsc):** Igual que el anterior pero con una comunicación ya establecida.

El método queda implementado de la siguiente forma:

```
byte atr[] = pcsc.getATR();
if (java.util.Arrays.equals(atr, ATR_NEXUS_ONE)) {
    //NO terminar,si no se pierde la comunicacion
    //pcsc.fterminar();
    return NEXUS_ONE;
} else if (java.util.Arrays.equals(atr, ATR_MIFARE1K)) {
    pcsc.fterminar();
    return MIFARE1K;
} else if (java.util.Arrays.equals(atr, ATR_TOPAZ)) {
    pcsc.fterminar();
    return TOPAZ;
} else if (java.util.Arrays.equals(atr, ATR_MIFARE_ULTRALIGTH)) {
    pcsc.fterminar();
    return MIFARE_ULTRALIGTH;
} else if (java.util.Arrays.equals(atr, ATR_NOKIA_6212)) {
    //NO terminar,si no se pierde la comunicacion
    //pcsc.fterminar();
    return NOKIA_6212;
}
pcsc.fterminar();
return -1;
```

Se comprueba uno a uno con los diferentes ATR almacenados. En el caso de que no se trate de un elemento pasivo, no hay que finalizar la comunicación en este punto ya

que da problemas en el otro extremo, al finalizar inesperadamente. En caso de que no sea reconocido devuelve "-1".

6.3.4 NFCException

Clase que hereda de Exception y sirve para representar las excepciones relacionadas con NFC. Tiene 2 métodos constructores:

- **NFCException()**: Excepción sin información
- **NFCException(String s)**: Excepción con información del motivo por el que ocurrió.

6.3.5 ISO14emu

Es la clase que sirve para la emulación de tarjetas. Permite la comunicación entre el lector y otro elemento activo como pudiera ser un teléfono móvil u otro lector. Los métodos son

- **ISO14emu(int debug):**Constructor que admite modo debug. Se queda a la espera de la aproximación de un teléfono móvil e inicia la conexión con el lector (con PCSC.iniciar()).
- **intercambio(byte tx[]):** Realiza la transmisión/recepción de datos. En este punto si se deseara realmente emular una smartcard, debería implementarse el protocolo que utilicen esta. Pero como solamente se le da el uso para el intercambio de información sin ningún formato concreto, se envían y reciben los datos tal cual.
- **terminar():**finaliza la comunicación de forma segura, esperando la retirada del teléfono móvil.

El método constructor es ligeramente diferente si desea emplear con un Samsung Nexus S o un Nokia 6212.En primer lugar, con el método iniciar() de PCSC se queda a la espera de la aproximación del teléfono. Cuando comienza la comunicación, se envían las instrucciones al lector que permiten que funcione en modo emulación:

```
nfc4 = new PCSCj();
nfc4.iniciar(debug);
nfc4.lectorctl(TgInitAsTarget);
nfc4.lectorctl(TgGetData);
nfc4.lectorctl(TgSetData);
if (!nokia) {
    nfc4.lectorctl(TgGetData);
    nfc4.lectorctl(TgSetData);
}
```

En el caso de los Nokia, en el inicio de la comunicación hay que añadir 2 APDUs.

El método intercambio se basa en lectorctl de PCSC. Es igual tanto para el Nokia como para el Nexus S. Los APDUs incluyen cabeceras determinadas por TgGetData para obtener los datos recibidos y TgSetData para responder con otros. Al estar realizando la emulación de una smartcard siempre es necesaria una recepción de datos seguido de una respuesta. Como se puede ver, primero se le envía la instrucción de TgGetData al lector y se extraen los datos del APDU. Después se le envía la instrucción TgSetData junto con los datos para realizar la respuesta.

```
byte ta1[] = new byte[]{(byte) 0xFF, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0xD4, (byte) 0x86};
ta1[4] = (byte) (ta1.length - 5);
byte rx2APDU[] = nfc4.lectorctl(ta1);
byte ta2[] = new byte[]{(byte) 0xFF, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0xD4, (byte) 0x8E};
byte tee[] = new byte[7 + adat.length];
System.arraycopy(ta2, 0, tee, 0, 7);
System.arraycopy(adat, 0, tee, 7, adat.length);
tee[4] = (byte) (tee.length - 5);
nfc4.lectorctl(tee);
if (rx2APDU.length < 6) {
    reset14();
    throw new NFCException("Se perdió la información");
}
byte rxAPDU[] = new byte[rx2APDU.length - 5];
System.arraycopy(rx2APDU, 3, rxAPDU, 0, rxAPDU.length);
return rxAPDU;
```

En caso de que la información recibida no tenga la longitud mínima será debido a que ha ocurrido algún problema.

Para finalizar la conexión es necesario añadir una espera de unos segundos. Esto es debido a que cuando se termina de transmitir, el lector está inactivo por uno instante. Durante ese instante, el lector no detecta ningún elemento, por lo que parecerá que se ha retirado el teléfono erróneamente y después se ha aproximado de nuevo. Añadiendo la espera, el lector puede volver a la actividad y comprobar si el teléfono sigue aproximado o se retiró ya.

```
try {
    Object o = new Object();
    synchronized (o) {
        //Tiempo de espera de unos 2600ms
        o.wait(2600);
    }
} catch (Exception ex) {
}
return nfc4.terminar();
```

6.3.6 SCard14emu

Se trata de una clase **obsoleta** que no debe utilizarse, ya que más difícil de implementar y no se comprueban los errores automáticamente (no lanza excepción en

esos casos). Se mantiene por compatibilidad hacia atrás con diferentes aplicaciones que hicieron uso de esta clase en anteriores versiones.

6.3.7 MifareException

Clase hereditaria de Exception para manejar las excepciones relacionadas con las tarjetas Mifare. Tiene 2 métodos constructores:

- **MifareException:** Para crear un excepción sin mensaje.
- **MifareException(String mens):** Para crear una excepción indicando el motivo.
-

6.3.8 MifareUltraligth

Clase para leer y escribir etiquetas Mifare Ultraligth.

- **MifareUltraligth(int debug):**Constructor que admite debug. Llama al método PCSC.iniciar() para realizar la detección y conexión.
- **rBloque(byte bloque):**Lee un determinado bloque(4 bytes)
- **wBloque(byte bloque,byte[] bloq):**Escribe en el bloque indicado los 4 bytes
- **getUID():** Sirve para obtener el identificador único de las etiquetas Mifare Ultralight.

El método constructor se limita a comenzar la conexión con PCSC como en el resto de los casos:

```
PCSCj pcsc = new PCSCj();
pcsc.iniciar(0);
```

El método lBloque(byte bloque) en primer lugar comprueba de que no se intente leer uno de los 4 bloques reservados por el fabricante. Si es un bloque correcto, se procede a enviar la instrucción para leer el bloque requerido. De la respuesta obtenida, se extrae el contenido de ese bloque.

```
if (bloque < 4) {
    System.out.print("Bloques del 0 al 4 reservados");
}
byte rd[] = new byte[]{(byte) 0xFF, (byte) 0xB0, (byte) 0x00,
(byte) 0x04, (byte) 0x04};
rd[3] = bloque;
byte[] rx = pcsc.lectorctl(rd);
byte rx2[] = new byte[rx.length - 2];
System.arraycopy(rx, 0, rx2, 0, rx.length - 2);
return rx2;
```

Para realizar la escritura de un bloque, se realiza de forma similar. Primero se realizan 2 comprobaciones: Que el tamaño del array de bytes se corresponda con el tamaño del sector, es decir 4, y que el número de bloque no pertenezca a los 4 primeros bloques reservados por el fabricante. Después se prepara el APDU con la instrucción de escritura, indicando la posición y los bytes a escribir.

```
if (datos.length != 4) {
    System.out.print("La longitud de los datos debe ser 4");
}
if (bloque < 4) {
    System.out.print("Bloques del 0 al 4 reservados");
}
byte wr[] = new byte[]{(byte) 0xFF, (byte) 0xD6, (byte) 0x00,
(byte) 0x04, (byte) 0x04, (byte) 0xAA, (byte) 0xBB, (byte) 0xCC,
(byte) 0xDD};
wr[3] = bloque;
System.arraycopy(datos, 0, wr, 5, 4);
byte[] rx = pcsc.lectorctl(wr);
```

Por último, el método getUID obtiene el identificador único de la etiqueta, enviando el instrucción correspondiente para este tipo de etiquetas y obteniéndolo directamente en la respuesta.

```
byte rx[] = pcsc.lectorctl(new byte[]{(byte) 0xFF, (byte) 0xCA,
(byte) 0x00, (byte) 0x00, (byte) 0x00});
byte UID[] = new byte[rx.length - 2];
System.arraycopy(rx, 0, UID, 0, rx.length - 2);
return UID;
```

6.3.9 Mifare 1k

Las tarjetas Mifare 1k, al ser más complejas que las Mifare UltraLight, tienen una cantidad de métodos mayor para realizar la autenticación previa. A diferencia del resto de las clases que hacen uso de PCSC, esta realiza los mimos procedimientos pero haciendo uso de las clases de *java.smartcardio* directamente.

- **Mifare1k(int debug):** Método constructor, al que se le indica la cantidad de información mostrada con el valor de debug. Espera que haya una tarjeta próxima al lector.
- **cargarClave(byte clave[], int tipo):** Método que prepara en el lector la clave a utilizar con la Mifare 1k.
- **autenticar(byte bloque):** Realiza la autenticación en el sector correspondiente a ese bloque, con la clave cargada anteriormente. Si la clave es correcta devuelve true.
- **rBloque(byte bloque):** Realiza la lectura de dicho bloque. Es necesario haber cargado la clave previamente y recomendable la autenticación.
- **wBloque(byte bloque, byte[] datos):** Escribe en el bloque.
- **cBloque(byte bloque):** Pone el bloque con todos los bytes a '0'.

- **cambiarClave(byte bloque, byte[] clave, int condiciones):** Cambia la clave de un determinado sector, con las condiciones de lectura,escritura y acceso representadas por el valor del int condiciones.
- **getUID():** Obtiene el identificador de la Mifare 1k.
- **cerrar():** Termina la conexión y espera la retirada de la tarjeta.

La implementación de estos pasos quedan reflejados de forma sencilla en el pauquete . Un ejemplo de código seria el siguiente:

```
byte[] clave = new byte[]{(byte) 0xFF, (byte) 0xFF, (byte) 0xFF,
(byte) 0xFF, (byte) 0xFF, (byte) 0xFF};

byte blq = 0x10;//Sector 4,bloque 0
boolean resultado = false;//90 00 true,63 00 false
Mifare1k m1 = new Mifare1k(0);
resultado = m1.cargarClave(clave, Mifare1k.TIPO_A);
System.out.println("Cargar clave:....." + resultado);
resultado = m1.autenticar(blq);
System.out.println("Autenticacion:...." + resultado);
resultado = m1.rTBloque(blq);
System.out.println("Prueba leer:......" + resultado);
m1.cerrar();
```

El anterior ejemplo crea el objeto tarjeta, y carga la clave en el lector. Después realiza la autenticación en el lector y hace una prueba de lectura. Adicionalmente tenemos métodos de lectura, de escritura y de cambio de claves. Para evitar problemas, está programado de tal forma que solo te permita escribir en los bloques libre, en los bloques reservados solo se puede acceder con el método de cambio de claves.

Internamente al crear el objeto Mifare, éste espera la aproximación de una tarjeta. Cuando la tarjeta es detectada los métodos de lectura/escritura/autenticación realizan el envío de APDUs al lector. Un ejemplo del método de lectura:

```
public byte[] rBloque(byte bloque) throws MifareException {

int n = (bloque + 1) % 0x04;
if (n == 0) {
    throw new MifareException("Numero de bloque(" + bloque + ")
reservado.");
}
//; [2] Authenticate sector 0, Block 0 with key at location 0
//FF 86 00 00 05 01 00 00 60 00 (9000)
byte[] cmd = new byte[]{(byte) 0xFF, (byte) 0x86, (byte) 0x00,
(byte) 0x00, (byte) 0x05, (byte) 0x01, (byte) 0x00, bloque, tipoK,
(byte) 0x00};
byte[] rx = this.rawtx(cmd);
if (debug > 1) {
```

```
    System.out.println(new String(toHex(rx)));
}
if ((byte) rx[rx.length - 2] == (byte) 0x63) {
    return null;
}
//; [3] Read the full 16 bytes from Sector 0, Block 2
//FF B0 00 01 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx]
(9000)
cmd = new byte[]{(byte) 0xFF, (byte) 0xB0, (byte) 0x00, bloque,
(byte) 0x10};
rx = this.rawtx(cmd);
if (debug > 1) {
    System.out.println(new String(toHex(rx)));
}
if (rx.length > 2) {
    byte[] rx2 = new byte[rx.length - 2];
    System.arraycopy(rx, 0, rx2, 0, (rx.length - 2));
    return rx2;
} else {
    return null;
}

}
```

En el primer paso se comprueba que no tratemos de leer un bloque reservado. En tal caso se lanza una excepción.

En la segunda parte, se prepara la APDU de autenticación y se envía al lector con rawtx. En caso de no realizarse con éxito, se recibe un 0x63 y entonces se evita la posterior lectura devolviendo null. Si se realizó la autenticación perfectamente (0x90), se procede al envío de la APDU de lectura, cuya respuesta contendrá los datos almacenados en el bloque solicitado.

6.3.10 PCSCjt y PCSCListener

Hasta este punto, todas las clases realizan su función en primer plano, dificultando su uso en caso de querer trabajar con otro hilo de ejecución principal y sin tener que gestionar hilos o *threads*.

Para ello se ha creado este paquete experimental, que comienza sentar las bases para el trabajo con el lector en segundo plano, trabajando con lo que se conoce como "eventos" o "alarmas" y clases que están a la escucha de estos avisos. La clase PCSCjt es la clase encargada de hacer los preparativos previos y PCSCListener es la que recibe el aviso. Los métodos en PCSCjt son similares a los anteriores de PCSC, salvo que no esperan la aproximación con el método iniciar. Comienza a estará la escucha en segundo plano cuando se llama al método addPCSCListener() como se puede ver en el siguiente fragmento:

```
public void addPCSCListener(PCSCListener pl) {
pcl = pl;
tlis = new Thread(new Runnable() {
```

```
public void run() {
    try {
        while (true) {
            // Espera el cambio de estado
            try {
                if (card == null) {
                    //
                    terminal.waitForCardPresent(0);
                    card = terminal.connect("*");
                    ch = card.getBasicChannel();
                    pcl.aproximacionT(PCSCjt.this);
                } else {
                    terminal.waitForCardAbsent(0);
                    card.disconnect(true);
                    card = null;
                    pcl.retiradaT();
                }
            } catch (CardException e1) {
                // Reintetamos conectar si esta presente.
                if (terminal.isCardPresent()) {
                    card = terminal.connect("*");
                    pcl.aproximacionT(PCSCjt.this);
                }
            }
        }
    } catch (CardException e2) {
        // Excepcion grave,salimos del bucle
        System.out.println("excepcion.finalizo");
    }
}
});
tlis.start();
}
```

Lo interesante es que ahora el código de la aplicación final quedara más claro y se ejecutara cada vez que se aproxime un elemento NFC. Gracias a la interfaz PCSCListener, que funciona de forma similar a los Listener y Eventos de java.awt, se pueden implementar los métodos que se ejecutaran cada vez que se aproxime y retire una tarjeta. Los 2 métodos son:

- **aproximacionT(PCSCjt pjs)**: Este método se ejecutara cuando se aproxime una tarjeta o teléfono móvil, y permite trabajar con el lector de igual manera que con la clase anterior PCSCj.
- **retiradaT()**: Cuando se retira la tarjeta, se llama a este método, por si es necesario tener conocimiento de este aviso.

6.3.11 SC14T y ListenerSC14

Estas dos clases de igual manera se encuentran en gammasol.pcsct porque trabajan en segundo plano. A diferencia de las 2 anteriores, que están diseñadas para trabajar con diferentes tipos de conexión, estas 2 están diseñadas para trabajar exclusivamente

como emulación de tarjetas y únicamente con el Nexus One. Al igual que las anteriores, son experimentales y aún están en desarrollo, por lo que es recomendable hacer uso ISO14emu.

La clase SC14T hace uso de las 2 anteriores (PCSCjt y PCSCListener).En este caso con el método addListenerSC14 añadimos la clase que estará a la escucha (internamente llama al anterior addPCSCListener). En cambio, esta clase solo es avisada cuando se conecta un Nexus One, y no avisa cuando se retira, por lo que únicamente posee el método conectado (SC14T sc14t).

6.3.12 El paquete de utilidades

Este paquete contiene 3 clases que aunque no son importantes conviene citar. La clase Info muestra es capaz de mostrar diferentes características del sistema, sin tener que profundizar en detalles. Los 3 métodos mostrarFirmware(), mostrarATR() y mostrarUID() relizan los citados anteriormente. El método propiedadesCPU() muestra diferente información del sistema que puede ser importante: El directorio de trabajo actual, el sistema operativo en el que se está ejecutando la aplicación y la arquitectura de la CPU.

La clase Util posee el método toHex() que facilita la visualización de un array de bytes devolviendo una cadena de texto con los correspondientes valores en hexadecimal. Es utilizado por multitud de clases que necesitan mostrar esta información.

La clase Cifrado posee los métodos encrypt y decrypt para cifrar y descifrar la información, para que sea transparente y puedan utilizarse diferentes métodos de cifrado en el futuro. Por defecto se hace uso del algoritmo AES/ECB/PKCS5Padding con claves de 256 bits.

6.4 Funcionamiento de la aplicación

En este punto ya está realizada una completa librería que permite trabajar con gran cantidad de las funcionalidades del lector. Ahora ya se puede pasar a tratar de conseguir los objetivos propuestos: Realizar una aplicación de autentificación. Recordamos brevemente los escenarios posibles:

- Un solo PC con una base de datos local. Los datos de usuario y la comprobación están en la misma máquina.
- Una red de ordenadores con una base de datos en red. Los datos del usuario están en el PC, pero la base de datos donde se realiza la autentificación están en un servidor.

- Una red de ordenadores con base de datos y ficheros de usuario en red. Este ejemplo podría aplicarse en un entorno como los PCs de la uc3m, en especial los del departamento de telemática, donde se accede a los ficheros de usuario (home) por NFS. Se le añadiría la autentificación de la forma en la que se realiza aquí.

De los 3 escenarios, por su sencillez, se realizara el primero. Pero con pocos cambios se podría llevar a cabo en los otros dos. Y eso no es más que un ejemplo de los diferentes usos que se le podrían dar: Cajeros, Control de acceso

6.4.1 Modo de uso

Para ello la aplicación en el terminal móvil dispondrá de 2 modos de funcionamiento: Registro y Login. El primero de ellos se encarga la primera vez de que el móvil sea registrado en el sistema. Posteriormente, la aplicación de Login permitirá a todo los usuarios con su terminal registrado acceder a su cuenta de usuario.

6.4.2 Protocolo de autentificación.

Se definen los siguientes protocolos de funcionamiento de la aplicación. En el caso de registro:

Móvil	PC
"RegistroMP----------"	"RegistroPC----------"
IMEI	…
..	ka
..	kb
"OK"	"OK"

Tabla 8: Protocolo del registro

En el primer intercambio se envían los identificadores de la conexión."RegistroMP" se envía desde el móvil para señalar que se trata de un teléfono móvil con la aplicación de registro preparada. En el otro extremo, el PC envía "RegistroPC" para indicar que se trata de un PC con la aplicación de registro. Ambos elemento deben comprobar que se trata de la aplicación correcta. Una vez comprobado, el PC envía las claves correspondientes que serán utilizadas en la aplicación de login. Serán guardadas en la memoria del dispositivo. Desde el móvil se envía el IMEI (o la dirección Bluetooth en caso de no estar firmada) como identificador del móvil. En este punto el PC guarda en su base de datos, el nombre de usuario para el login, el IMEI que le corresponde y las 2 claves que sirven para autenticarse. Finalmente si todo ha ido bien se envía un mensaje "OK" para finalizar la comunicación.

En el caso de la aplicación de la aplicación de Login:

Movil	PC
"IdentificacionMP-----"	"IdentificacionPC-----"
...	RAND(ka) / "NO"
RAND(kb)	...
...	"OK" / "NO"

Tabla 9: Protocolo del login

Como se ve el protocolo es bastante seguro. Aunque se escuche el IMEI/Dirección bluetooth, se deben conocer las claves. Y en caso de copiar los archivos internos donde están las claves a otro terminal, al utilizar esta aplicación se detectaría otro IMEI diferente por lo que no serviría. En todo caso, para aplicaciones de seguridad extrema, las aplicaciones se desarrollan para el elemento seguro (Secure Element) normalmente dentro de la SIM e inaccesible desde el exterior. Para instalar aplicaciones en esta parte las aplicaciones deben estar firmadas por lo que dificulta enormemente el robo de información.

6.5 Configuración Linux

Ahora hay que integrar la aplicación anterior dentro de una maquina Linux, de tal forma que quedara transparente en caso de realizar futuras versiones. El interés de la aplicación está en la autenticación del usuario y a partir de esta parte se toma eses usuario y un servicio se encargara del login. Las partes necesarias a implementar son la base de datos, los script de arranque y la configuración de usuario.

6.5.1 Base de Datos

La base de datos utilizada está basada en PostreSQL, que permite la gestión y manipulación fácilmente con un driver desde la aplicación en Java.

Para la aplicación se utiliza una tabla llamada usuarios. Esta tabla dispone de 4 columnas que permiten guardan los datos necesarios de los usuarios. Las cuatro columnas son “identificador”, “login”, “claveb1” y “claveb2”. La columna “identificador” almacena el valor del IMEI o de la dirección Bluetooth del dispositivo móvil. En caso de que se utilice una Mifare 1k, se corresponde con el UID de la tarjeta. La columna “login” contiene el nombre de usuario que se utiliza para el login. Puede repetirse, ya que una misma persona podría tener uno o más móviles. Las dos últimas columnas contienen las claves de acceso. Deben coincidir con las almacenadas en el móvil o tarjeta.

6.5.2 Servicio

Para que automáticamente se inicie todo el proceso de autenticación, se deben añadir el siguiente archivo a los script de arranque:/etc/init.d/serviciologinnfc.

Este archivo se encargara del proceso de arranque junto al sistema. Es necesario añadirlo al runlevel 2 para que se inicie cuando el sistema, ha iniciado el resto de los servicios necesarios (tales como pcscd, demonio que identificara al lector y permitirá la comunicación con el).Para esto solo es necesario añadir un enlace a /etc/rc2.d,como se ve con el siguiente comando:

sudo ln -s /etc/init.d/serviciologinnfc /etc/rc2.d/S80serviciologin

De esta manera nos aseguramos que todos los pasos siguientes se realizaran automáticamente al arrancar el sistema. Este script comenzara automáticamente le sesión del usuario loggernfc cuyo único cometido será ejecutar la aplicación que lee el lector e iniciar la sesión de los demás usuarios. El script que se encargue de estas 2 cosas se llama scrpitNFC. La descripción de este script es la siguiente:

```
# nerr : Numero de errores.

# varJAVA : Salida programa java
# varUsuario : Usuario de login
#sudo pcscd
sleep 8
nerr=0;
while [ "$nerr" -lt "8" ]
do
varJAVA="zxczcz"
varJAVA=`bash /home/fermio/variableNFC`
nc=`echo $varJAVA ----- | grep "NFCLOG" -c`
if [ "$nc" -eq "1" ] ; then
varUsuario=`echo $varJAVA | awk '{print $2}' FS=":"`
echo $varUsuario
sudo service gdm stop
sleep 2
sudo login -f $varUsuario
sudo service gdm start
#echo fin-----------
elif [ "$nc" -ne "1" ] ; then
echo error
let nerr=$nerr+1
sleep 3
fi
varJAVA=""
varUsuario=""
done
```

En primer lugar realiza una espera de 8 segundos, para asegurarse que arranco el servicio pcscd y detecto e inicializó el lector correctamente. El siguiente paso es inicializar la variable del número de errores. Como seguridad, en caso de haber más de 8

errores se finalizara el script. Se implementa esta comprobación para evitar un bucle infinito con errores, bien por algún problema de java o del servicio pcscd, como probablemente pudiera ser que no se hubiera conectado ningún lector NFC. El siguiente paso es ejecutar la aplicación java. Esta se limitaba a esperar un intento de autenticación por algún usuario, de modo que al finalizar utilizaremos la salida estándar para saber el usuario que logro autenticarse. El formato de salida es el siguiente NFCLOG:usuario, por lo que se comprueba que exista alguna línea con tal formato. En caso de no ser así, por algún error, se repetirá el proceso hasta alcanzar el número de errores máximo permitido (8 en este caso).Si la ejecución finalizo con éxito, podremos saber el usuario que se autenticó y de esta manera iniciar su sesión.

6.5.3 Usuario

Para permitir a los usuarios logearse por NFC hay que preparar lo siguiente:

En primer lugar añadir los usuarios al grupo loginnfc. De esta manera la aplicación de registro lo reconocerá como tal.

Después hay que añadir al archivo .profile las siguientes líneas. Estas se encargan de comprobar si se ha lanzado previamente algún servidor gráfico y en caso de no ser así (como es el caso de login por NFC) se arrancara al inicio de la sesión.

```
varstadogdm=`service gdm status | grep -c waiting`
if [ "$varstadogdm" -eq 1 ] ; then
(startx) && exit 0
fi
```

En primer lugar se comprueba que el servidor grafico está en la pantalla inicial de login. Si es así, es que se ha iniciado sesión internamente por línea de comandos, por lo que es necesario arrancar el servidor gráfico. Al finalizar este también finalizara la sesión automáticamente. Si no es así, es que el usuario introdujo su contraseña por pantalla, y no hay que hacer nada.

6.6 Aplicación móvil

Hasta este punto, toda la descripción ha sido referente a la aplicación desde el lado del PC. La otra parte corresponde con los terminales móviles y las 2 plataformas sobre las que se ha trabajado son J2ME y Android. Las aplicaciones deben realizar la comunicación NFC, implementar el protocolo de autenticación y almacenar las claves de acceso. Las aplicaciones se dividen en 2 partes la de Registro en el sistema que obtiene las claves de acceso y las almacena y la de Identificación que hace uso de las claves almacenadas.

6.6.1 J2ME

De cara al usuario habrá 3 iconos que presentaran la información de forma sencilla. Los posibles resultados son transacción con éxito, error de comunicación y problema de transacción. Los iconos son un tick verde, una exdamación y un icono de negación respectivamente.

6.6.1.1 Registro

Cuando se ejecuta la aplicación por primera vez, aparece la pantalla de bienvenida de registro. Para proceder con el registro se pulsa el botón central y se aproxima el móvil al lector. La pantalla inicial es de la siguiente forma:

Ilustración 14: Captura inicio de registro

Si todo ha ido bien, la comunicación se realiza con éxito y se añade al usuario a la base de datos, la pantalla mostrara la siguiente confirmación:

Ilustración 15: Captura registro con éxito

En caso de que las claves no sean correctas:

Ilustración 16: Captura no registrado

Y por último, si ocurre algún error durante la comunicación aparece la pantalla de advertencia con el posible motivo del error.

Ilustración 17: Captura error en el registro

El código que consigue este funcionamiento es el siguiente. En primer lugar, para que la aplicación se inicie en modo registro o modo login comprobara si están las claves guardadas. En el caso de estarlo iniciará el modo login en la caso de que, el modo registro.

```
RecordStore rs = null;
//Comprobamos si ya existen las claves
try {
    rs = RecordStore.openRecordStore("ClaveB2", false);
    byte data[] = rs.getRecord(1);
    rs.closeRecordStore();
} catch (RecordStoreNotFoundException rsnf) {
    //Salta si no existe,cerramos antes por si acaso
    try {
        rs.closeRecordStore();
    } catch (Exception e) {
        System.out.println("err2");
    }
    modoRegistro();
    return;
} catch (Exception e) {
    System.out.println("err1");
}
//Si existen las claves,pasamos al modo Login
modoLogin();
al modo Login
modoLogin();
```

El método modoRegistro() prepara la pantalla y la comunicación para registrar el dispositivo en la base de datos, implementando el protocolo correspondiente.

```
ISO14443Connection smc = (ISO14443Connection)
Connector.open(tmp.getUrl(ISO14443Connection.class));
byte rx[] = null;
byte cB1[] = null;
byte cB2[] = null;
try {
    //envio peticion
    byte identf[] = "RegistroMP----------".getBytes();
    byte rx2[] = smc.exchangeData(identf);
    if (!"RegistroPC----------".equals(new String(rx2))) {
        rx = rx2;
        smc.close();
        form.set(2, new StringItem("", "NO es la aplicacion
login"));
        form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
        return;
    }
    //envio identificador

smc.exchangeData(javax.bluetooth.LocalDevice.getLocalDevice().getB
luetoothAddress().getBytes());
    //recibo claveB1
    cB1 = smc.exchangeData(dummy);
    //recibo claveB1
    cB2 = smc.exchangeData(dummy);
    //recepcion respuesta
    rx = smc.exchangeData("OK".getBytes());
    smc.close();
    //smc = null;
} catch (Exception e) {}
```

En primer lugar se crea una conexión ISO14443Connection que es la que permite el envío y recepción de datos de una smartcard, o de la smartcard emulada en el lector. Una vez establecida la conexión, se envía el identificador de la aplicación de registro, y espera recibir el propio identificador del lector. En caso de no ser así, se cierra la conexión y se muestra el mensaje de *warning* por pantalla.

Si se trata de la aplicación correcta, se envía el identificador del móvil, que se trata de la dirección Bluetooth (ya que para obtener el IMEI se necesita firmar la aplicación). En los siguientes pasos se reciben las claves para posteriormente guardarlas en un registro del móvil. Desde el móvil no se envía información. En el último envío, se confirma que todo ha ido bien.

Tras finalizar de la comunicación se comprueba que todo ha salido bien. Los últimos datos recibidos confirmaran si todo se ha ejecutado correctamente, hubo algún problema de conexión y no se pudo guardar el usuario por algún motivo.

```
if (rx == null) {
    form.set(2, new StringItem("", "Problema de conexion"));
```

```
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
} else if ("NO".equals(new String(rx))) {
    form.set(2, new StringItem("", "NO Registrado"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/block.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
} else if ("OK".equals(new String(rx))) {
    form.set(2, new StringItem("", "Registro OK"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/accept.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
    guardarClaves(cB1, cB2);
} else {
    form.set(2, new StringItem("", "Problema de conexion
desconocido"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));

}
```

6.6.1.2 Login

Una vez el terminal ha sido registrado, al abrir la aplicación aparece automáticamente la pantalla de login. Si por algún motivo, hubiera que borrar los datos guardados, se puede hacer desde el menú de opciones. De esta manera se podrá volver a la aplicación de registro.

Ilustración 18: Captura inicio de login

Al realizar la aproximación al lector, si todo ha ido bien y se realiza la autenticación con éxito, un tick verde confirmará que todo fue correcto.

Ilustración 19:Captura login con éxito

En el caso de que las claves no se correspondan con las almacenadas en la base de datos, fallará el proceso y mostrará la siguiente información:

Ilustración 20:Captura problema con el login

Por último, de igual manera que en la pantalla de registro, en caso de haber algún problema se muestra el icono de error con el posible motivo.

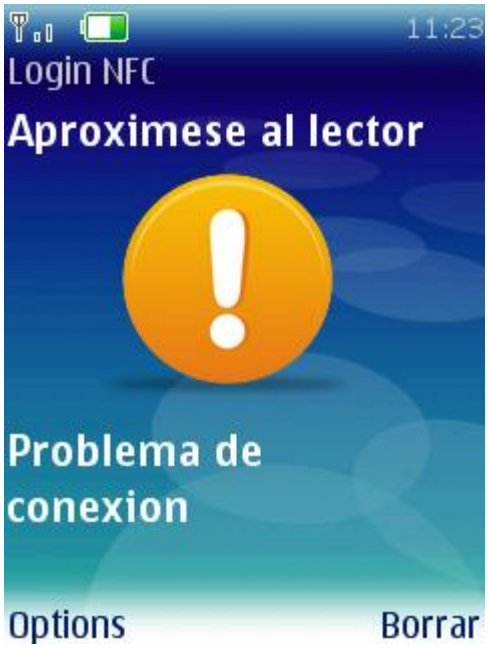


Ilustración 21:Captura error de conexión

En el caso del login, la implementación del protocolo queda de la siguiente manera:

```
ISO14443Connection smc = (ISO14443Connection)
Connector.open(tmp.getUrl(ISO14443Connection.class));
byte rx[] = null;
try {
    //envio peticion
    byte rx3[] = smc.exchangeData("IdentificacionMP-----
".getBytes());
    if (!"IdentificacionPC-----".equals(new String(rx3))) {
        rx = rx3;
        smc.close();
        form.set(2, new StringItem("", "NO es la aplicación de
registro"));
        form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
        return;
    }
    //envio identificador
    byte identf[] =
javax.bluetooth.LocalDevice.getLocalDevice().getBluetoothAddress()
.getBytes();
    smc.exchangeData(identf);
    //recibo RAND
    byte rx2[] = smc.exchangeData(dummy);
    if ("NO".equals(new String(rx2))) {
        rx = rx2;
        smc.close();
        form.set(2, new StringItem("", "NO Registrado en DDBB"));
        form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/block.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
        return;
```

```
        }
        Cifrado cf1 = new Cifrado(claveB1);
        byte[] randdescf = cf1.decrypt(rx2);
        Cifrado cf2 = new Cifrado(claveB2);
        //envio RAND
        smc.exchangeData(cf2.encrypt(randdescf));
        //recepcion respuesta
        rx = smc.exchangeData("OK".getBytes());
        smc.close();
    } catch (Exception e) {
    }
```

Primero se crea una conexión ISO14443Connection. En el primer intercambio, se envía y recibe el identificador de la aplicación. Si se trata de la aplicación correcta, se envía el identificador registrado (dirección Bluetooth).En el siguiente paso se recibe un número aleatorio cifrado, cuya clave se ha almacenado anteriormente. Se descifra y se vuelve a cifrar con la otra clave, para que el PC compruebe la autenticidad del móvil.

Y de nuevo, en función de cómo haya terminado la comunicación se mostraran los diferentes resultados citados anteriormente.

```
if (rx == null) {
    form.set(2, new StringItem("", "Problema de conexion"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
} else if ("NO".equals(new String(rx))) {
    form.set(2, new StringItem("", "NO Registrado correctamente"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/block.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
} else if ("OK".equals(new String(rx))) {
    form.set(2, new StringItem("", "Identificado correctamente"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/accept.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));
} else {
    form.set(2, new StringItem("", "Problema de conexion desconocido"));
    form.set(1, new javax.microedition.lcdui.ImageItem("",
Image.createImage("/warning.png"),
javax.microedition.lcdui.ImageItem.LAYOUT_CENTER, "OK"));

}
```

6.6.2 Android

De igual manera funciona la aplicación desarrollada para Android. Dispone de los 2 mismos modos, Registro y Login que se ejecutaran automáticamente al detectar si ya se ha registrado previamente el teléfono.

Las tres mismas imágenes servirán para visualizar de forma rápida el resultado de la autenticación.

6.6.2.1 Registro

El modo de funcionamiento es exactamente el mismo que en el caso de J2ME.

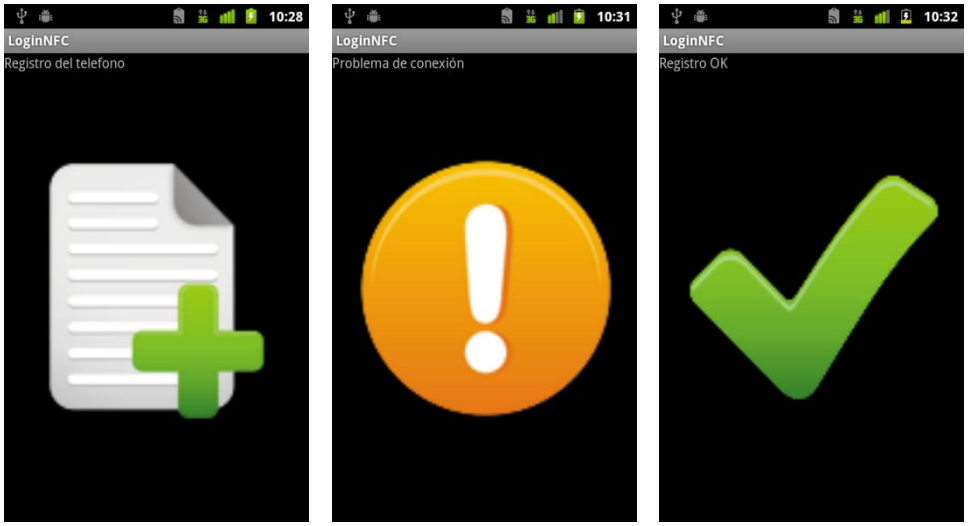


Ilustración 22: Capturas de registro en Android

El modo de funcionamiento es completamente el mismo, solo cambia la forma en que Android trabaja con NFC (Intent en lugar de Eventos) y la clase encargada de la lectura de smartcard (Isodep en lugar de ISO14443Connection). La clase Cifrado sigue siendo la misma, por lo que el cifrado es idéntico.

Las claves recibidas se guardan en un archivo de configuración para la aplicación.

6.6.2.2 Login

En el caso del login, también funciona de igual manera, pero tiene la ventaja que en Android se puede configurar para que se inicie la aplicación automáticamente al aproximarse al lector.

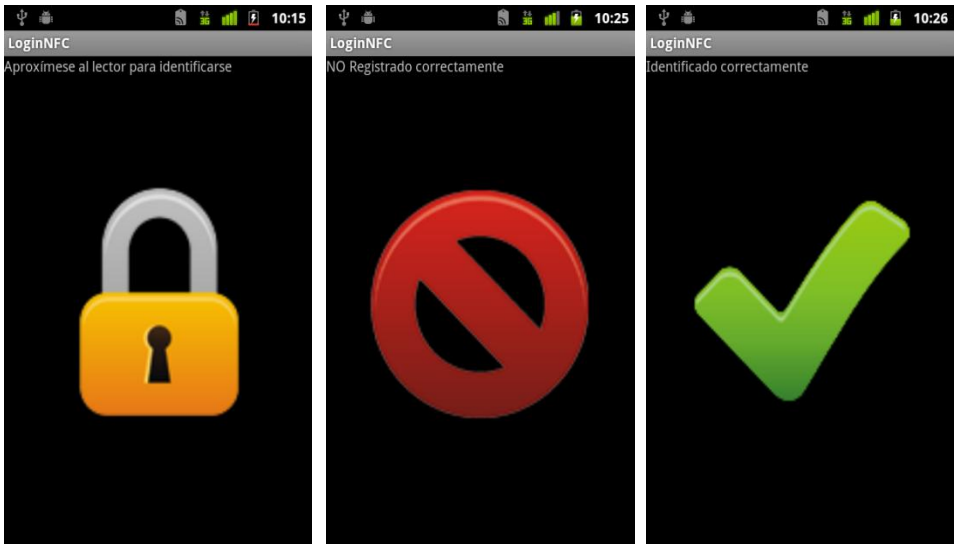


Ilustración 23: Capturas de login en Android

7 Pruebas

Para verificar el correcto funcionamiento de la aplicación es necesario acabar con éxito un conjunto de pruebas que cubra el mayor tipo de situaciones posibles. Se comienza con la base de la aplicación.

En primer lugar se realizan pruebas de lectura escritura en diferentes bloques de un Mifare Ultralight. Se debe comprobar que se ha leído y escrito con éxito. Se prueba sobre diferentes tarjetas varias veces y el resultado es satisfactorio.

En siguiente lugar se realizan las mismas operaciones sobre una Mifare 1k.Se comprueba que no se permite realizar estas operaciones sobre los bloques reservados. Después se prueba a cambiar las condiciones de acceso y la contraseña. Igualmente con diferentes tipos de tarjeta y varias veces.

En tercer lugar se comprueba el funcionamiento como emulación de tarjetas. Para hacer la prueba, se realiza en el envío y recepción de 4 cadenas de texto. Se preparan los 2 diferentes móviles que vamos a utilizar. Se comprueba varias veces, para comprobar el funcionamiento de la conexión y desconexión.

Por último, se realiza una aplicación en la que conviven las diferentes opciones: se detecta el tipo de elemento aproximado y en función de este se realiza la operación pertinente.

Una vez se tiene constancia de que la primera parte funciona correctamente, se puede pasar al realizar el siguiente conjunto de pruebas que corresponden a la aplicación

Se comprueba que el registro e identificación en la base de datos funciona correctamente. Se toma un teléfono móvil, se comprueba que el registro funciona correctamente, que se puede borrar y se puede registrar de nuevo. También se comprueba si se pueden eliminar los datos.

En el siguiente paso se comprueba que puede funcionar tanto con teléfonos móviles como las tarjetas. Se prueba a registrar, identificar, e igualmente borrar los datos contenidos en la tarjeta.

Por último, comprobar que se ha integrado el servicio correctamente en el sistema. Se comprueba el arranque y la ejecución. Para demostrar la convivencia con la pantalla de login del sistema operativo, se prueba acceder tanto por contraseña como por el móvil previamente registrado.

Llegados a este punto, se ha podido comprobar que las acciones que se realizaran en el uso cotidiano han sido correctamente probadas. Aun así es posible que el algunas situaciones excepcionales se produzca un error, el testeo de aplicaciones es un proceso largo que no puede asegurar al 100 % el correcto funcionamiento de una aplicación.

8 Conclusiones

A lo largo de este proyecto son 2 las necesidades que se han cubierto: La obtención de una herramienta para poder realizar multitud de aplicaciones con NFC en el PC, y el desarrollo de un ejemplo de implementación de un proceso de autenticación, entre PC y teléfonos móviles.

La librería NFC desarrollada para el lector ACR122 permite que cualquier nuevo usuario que desee realizar una aplicación NFC tenga una base bastante trabajada y documentada, que le permita implementar sus ideas. Las facilidades que se han logrado son muchas. Una de las más importantes que al estar desarrollada en Java es compatible con diferentes sistemas operativos (Windows y Linux), además de abrir la posibilidad de desarrollar aplicaciones web en forma de applet (como otras aplicaciones utilizadas por bancos, ministerios u organismos del estado que hacen uso del DNIe y lectores de tarjetas). La cantidad de protocolos implementados, permite la utilización de tarjetas y teléfonos móviles sin tener la necesidad de trabajar con APDUs. E incluso la posibilidad de utilizar un cifrado básico de la información que tratan las aplicaciones.

La aplicación de login sirve como ejemplo para demostrar el alcance y potencial de las aplicaciones que se pueden realizar trabajando con NFC. Permite ver cómo se puede integrar en un PC una forma de acceso alternativa a la dásica forma con usuario y contraseña u otras más actuales como lectores de tarjetas de contacto o dispositivos USB. El ejemplo del acceso a una cuenta de usuario es solo uno de los usos más comunes que se pueden realizar, pudiéndose extender a otros como acceso a edificios, cajeros. También se facilita la gestión almacenando los usuarios en una base de datos donde se guardan los nombres y las claves, con la posibilidad de trabajar con información personal adicional.

Trabajos futuros

Sobre la librería del lector hay aún trabajo por hacer, pues permite ser utilizado de más formas. Entre las posibles mejoras se encuentran las siguientes.

- **Identificador del lector**: Hasta ahora el móvil ha detectado el lector con un mismo identificador, pues no ha sido necesario utilizar algo así en ningún momento. Pero el lector permite cuando emula una tarjeta tener un identificador personalizado, que podría ser de utilidad en otro tipo de aplicaciones que tuvieran que distinguir entre lectores.
- **Varios lectores en un mismo PC**: Igualmente no ha sido necesario esta funcionalidad, pero con Java se puede trabajar perfectamente con distintos lectores a la vez. En un escenario en que podría ser por ejemplo un cine o servicio de transporte público, en el que se utilizara un lector para sacar y pagar entradas y otro para validarlas y acceder.
- **Envío de mensajes NDEF**: Es posible enviar mensajes NDEF desde el lector al Nexus S, que facilita la lectura de información en el caso de que se quiera

transmitir una información del tipo que suele utilizarse en los mensajes NDEF (URLs, imágenes, información de contacto,..). No existe mucha documentación al respecto, pero los desarrolladores del proyecto libre Libnfc ya lo han conseguido con éxito.

En cuanto a la aplicación los puntos a mejorar son los siguientes:

- **Implementación en una red local**: Se deja la posibilidad abierta para que puede utilizarse en este escenario, ya que la información de los usuarios se almacena en una base de datos.
- **Otros sistemas operativos**: Los sistemas operativos GNU/Linux facilitan la personalización y modificación de éstos. Y aunque en sistemas operativos Windows es mucho más complicado, es algo perfectamente realizable con bastante más trabajo.
- **Seguridad**: La aplicación presenta un nivel de seguridad acorde con el uso que se le está dando. Para utilizarlo en ámbitos más sensibles (bancos) había que mejorar este punto.

9 Diagrama de Gantt

La planificación de las tareas está orientada a semanas. La duración estimada de cada una de ellas es la que se muestra en la siguiente tabla:

Semanas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Aprendizaje J2ME y Android																
Aprendizaje NFC en J2ME Android																
Búsqueda de un lector apropiado																
Programación del lector																
Comunicación entre el lector y los terminales																
Desarrollo de la aplicación																
Implementación en SO																
Pruebas																

Tabla 10: Diagrama de tareas

Algunas de las tareas están solapadas, puesto que a medida que se están finalizando unas, se puede comenzar con las siguientes de forma paralela.

La cantidad de horas semanales se corresponde con el número de horas de una semana laboral, la cual consta de 40 horas de trabajo semanales. En caso el caso de las actividades que se solapan en una misma semana, se dividirán las horas de trabajo entre las dos tareas.

10 Presupuestos

El coste todos los componentes necesarios, ordenadores, teléfonos móviles documentación se ve reflejado en la siguiente tabla:

Material	Unidades	Precio	Coste
Ordenador	1	850	850
Lector ACR122	1	80	80
Mifare 1k	1	20	20
Nokia 6212NFC	2	150	300
Nexus S	1	550	550
Impresión documentación	-	-	50
Total			1.830

La posibilidad de utilizar entornos de desarrollo libres, y de la disponibilidad de documentación online tanto por proyectos libres como por los fabricantes, ayuda en gran medida a la reducción del coste.

Para realizar el cálculo de los costes de personal se tendrá en cuenta el salario aproximado de un Ingeniero Técnico de Telecomunicaciones. Sabiendo que el salario medio es de unos 1.600 €, la hora de trabajo (40 semanales, 160 mensuales) sale a 10 €.

Coste de personal		
Tareas	Horas	Coste
Aprendizaje J2ME y Android	120	1200
Aprendizaje NFC en J2ME y Android	80	800
Búsqueda de un lector apropiado	20	200
Programación del lector	60	600
Comunicación entre el lector y los terminales	140	1400
Desarrollo de la aplicación	160	1600
Implementación en SO	60	600
Pruebas	20	200
Total	640	6.400

El coste total viene dado por la suma de ambas partes, lo cual resulta en un presupuesto de **8.230 € (ocho mil doscientos treinta euros)**.

11 Bibliografía

WEBS

[1]Java Community Process
http://jcp.org/en/home/index

[2] NFC Forum: Technical Specifications
http://www.nfc-forum.org/specs/spec_list/

[3] NFC Forum: About NFC
http://www.nfc-forum.org/aboutnfc/

[4] Libnfc
http://www.libnfc.org/

[5] Recursos NFC de Nokia
http://wiki.forum.nokia.com/index.php/Category:Near_Field_Communication_%28NFC%29

[6] J2ME Technology Overview
http://www.oracle.com/technetwork/java/javame/java-me-overview-402920.html

[7] Página Oficial de lector ACR122
http://www.acs.com.hk/index.php?pid=product&id=ACR122U

[8] Android
http://www.android.com/
Android Developer Guide
http://developer.android.com/guide/index.html
Android Plataform Versions
http://developer.android.com/resources/dashboard/platform-versions.html
Android SDK info
http://developer.android.com/sdk/index.html

NOTICIAS Y OTROS ENLACES

[9] DNI electrónico en el móvil
http://www.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/DNI_movil

[10] NFC and WIFI
https://market.android.com/details?id=gammasol.nfcandwifi

[11] Renfe y NFC
http://www.xatakaon.com/tecnologia-de-redes/el-nfc-llega-a-espana-de-la-mano-de-vodafone-y-renfe

[12] Google Wallet
http://googleblog.blogspot.com/2011/09/launching-google-wallet-on-sprint-and.html

[13] Paypal y NFC
https://www.thepaypalblog.com/2011/07/paypal-uses-nfc-to-make-peer-to-peer-payments-easier-than-ever/

[14] Productos NFC de Gamma Solutions
http://gammasolutions.es/index.php?option=com_content&view=article&id=94&Itemid=229

Última consulta de todos los enlaces (7-Octubre-2011)

DOCUMENTOS

[15] Funcionamiento del chip PN532
AN10609_3, PN532 C106 Application Note (Rev. 1.2 – January 5, 2010).

[16] Funcionamiento del lector ACR122
ACR122 NFC Reader API

[17] Tarjetas Mifare 1K (Philips Semiconductors)
Mifare 1k Standard Card IC MF1 IC S50 (Rev. 1.5 – May 2001)

[18] Mifare Ultraligth (Philips Semiconductors)
Contactless Single-trip Ticket IC MF0 IC U1 (Rev 3.0 – March 2003)

[19] Arquitectura NFC
NFC Technology Overview, Jonathan Main (September 2009)

[20] NFC Data Exchange Format (NDEF)
NFC Forum, NDEF 1.0,NFCForum-TS-NDEF_1.0 2006-07-24

[21] Smart Poster Record Type Definition
NFC Forum, SPR 1.,1 NFCForum-SmartPoster_RTD_1.0 2006-07-24

[22] Text Record Type Definition
NFC Forum, RTD-Text 1.0, NFCForum-TS-RTD_Text_1.0 2006-07-24

[23] URI Record Type Definition
URI Record Type Definition

[24] Java a Tope: J2ME
Autores: Sergio Gálvez Rojas, Lucas Ortega Díaz

[25] Diseño e implementación de un prototipo para control de acceso de personas aplicando la tecnología NFC por medio del uso de teléfonos celulares compatibles con esta tecnología.
Escuela Politécnica Nacional (Ecuador)
Autores: Veloz Chérrez, Diego Fernando

IMÁGENES

- **Las siguientes ilustraciones provienen del documento[19]:**
  Ilustración 3: Card Emulation Mode
  Ilustración 4: Peer-to-Peer Mode
  Ilustración 5:Reader/Writer Mode
- **La siguiente ilustración priviene del documento[17]:**
  Ilustración 6: Esquema Mifare1k
- **La siguiente ilustración proviene del documento[18]:**
  Ilustración 7: Esquema Mifare Ultraligth

TABLAS

- **Las siguientes tablas provienen del documento[23]:**
  Tabla 3: Ejemplo de registro NDEF tipo URI
- **Las siguientes tablas provienen del documento[21]:**
  Tabla 4: Ejemplo de registro NDEF tipo Smartposter
- **Las siguientes tablas provienen del documento[22]:**
  Tabla 2: Ejemplo de registro NDEF tipo texto

12 Anexos

12.1 Términos

Término	Descrpcion
Android	Sistema operativo para smartphones desarrollado por Google
APDU	(Application Protocol Data Unit) unidad de comunicación entre tarjetas inteligentes y un lector de éstas. Definido en el ISO/IEC 7816.
ATR	Answer To Reset, información referente a la comunicación tarjeta-lector, estructura de datos, protocolo de transmisión, etc.
Check-in	Término utilizado en las redes sociales para determinar el proceso por el cual un usuario puede compartir donde se encuentra en un determinado momento al resto de usuarios de la red
Dalvik	Máquina virtual utilizada en dispositivos Android para gestionar las aplicaciones.
Header	Archivo de cabecera en C donde se declaran las funciones
iOS	Sistema operativo de los dispositivos móviles a Apple.
J2ME	Tecnología de tarjetas sin contacto propiedad de NXP Semiconductors
Libnfc	Proyecto libre que permite utilizar lectores NFC y desarrollar aplicaciones
Mifare	Tarjetas desarrolladas
NDEF	NFC Data Exchange Format (NDEF), formato para la transmisión de datos en NFC
Payload	Carga, contenido de un campo especificado en un protocolo.
RTD	Record Type Definition, especificación del tipo de registro almacenado en un mensaje NDEF
SmartCard	Tarjeta inteligente
Smartphone	Teléfono inteligente
SO	Sistema Operativo
URI	Uniform Resource Identifier, cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, etc.)
URL	Uniform Resource Locator, secuencia de caracteres que especifica un determinado recurso en Internet.
Webkit	Motor de renderizado web libre en el que están basados navegadores como Chrome, Safari o Konqueror
XML	Extensible Markup Language, Lenguaje de Etiquetado Extensible.